

UNIVERSIDAD DE CUENCA

FACULTAD DE INGENIERÍA

ESCUELA DE ELECTRÓNICA Y TELECOMUNICACIONES



"DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA PROTOTIPO DE GESTIÓN DE ACCESO A LAS AULAS DE LA FACULTAD DE INGENIERÍA MEDIANTE LLAVES ELECTRÓNICAS"

Tesis previa a la obtención del Título de:
Ingeniero en Electrónica y Telecomunicaciones

AUTOR:

Christian Xavier Arias Ordóñez

DIRECTOR:

Ing. Remigio Clemente Guevara Baculima

CUENCA – ECUADOR

2015

RESUMEN

El objetivo principal de ésta tesis es diseñar un prototipo electrónico con conexión inalámbrica a Internet, que permita administrar de forma automática el acceso a las aulas y laboratorios de la Facultad de Ingeniería de la Universidad de Cuenca. Para lograrlo se ha diseñado el dispositivo de forma que en conjunto con un lector RFID, se utilicen tarjetas magnéticas como llaves electrónicas. De igual manera una aplicación creada para smartphones con sistema operativo Android, funcionará también como llave electrónica, permitiendo escoger al usuario la opción que preferiría utilizar en caso de implementar en algún momento el prototipo. La administración de los dispositivos así como de la base de datos de docentes, se la puede realizar a través de una aplicación web diseñada específicamente para ésta tesis y que reside en un servidor HTTP. Finalmente la comunicación entre el servidor y el prototipo electrónico es posible a través de un conjunto de funciones o API desarrollada en PHP.

Palabras clave: conexión inalámbrica, prototipo, RFID, aplicación móvil, Android, llave electrónica, aplicación web, servidor HTTP, API, PHP.

ABSTRACT

The main objective of this thesis is to design an electronic prototype with wireless Internet access, allowing automatically manage access to classrooms and laboratories of the Engineering Faculty of the University of Cuenca. To achieve this, a device was designed so that together with an RFID reader, magnetic cards can be used like electronic keys. On the other hand an application designed for smartphones with Android operating system, will also function as electronic key, allowing the user to choose the option that prefers to use in case that in the future the prototype is implemented in the faculty. The device management as well as the database of professors, can be managed through a web application designed specifically for this thesis and that resides on a HTTP server. Finally communication between the server and the electronic prototype is possible through a set of functions or API developed in PHP.

Key words: electronic prototype, wireless, RFID, mobile application, Android, electronic key, web application, HTTP server, API, PHP.



CONTENIDO

INTRODUCCIÓN	13
Presentación	13
Justificación.....	13
Alcance	14
Objetivo General	15
Objetivos Específicos	15
CAPÍTULO I	16
1. MARCO TEÓRICO	16
1.1. INTRODUCCION.....	16
1.2. RFID: IDENTIFICACION POR RADIO FRECUENCIA.....	16
1.2.1. Descripción y funcionamiento de un sistema RFID.....	16
1.2.2. Etiquetas.	18
1.2.3. Frecuencias de operación.....	18
1.3. INTERFAZ SPI	19
1.3.1. Funcionamiento	19
1.4. ARQUITECTURA CLIENTE – SERVIDOR	20
1.5. MODELO TCP/IP.....	22
1.5.1. Aplicaciones TCP/IP	22
1.5.2. Puertos y Sockets.....	23
1.6. PROTOCOLO HTTP	25
1.6.1. Solicitudes HTTP.....	25
1.6.2. Respuestas HTTP	26
1.6.3. URL	27
1.7. SISTEMA GESTOR DE BASE DE DATOS.....	27
1.7.1. Componentes de los SGBD	28
1.7.2. Modelos de datos	28
1.7.3. Modelo Entidad – Relación	29
1.8. LENGUAJE SQL	31
1.8.1. Elementos del lenguaje	32
1.9. BASE DE DATOS RELACIONAL	32
1.9.1. Claves	33



1.9.2. Restricciones.....	33
CAPÍTULO II	34
2. DISEÑO DEL SISTEMA	34
2.1. DIAGRAMA DEL SISTEMA	34
2.2. DESCRIPCIÓN Y FUNCIONAMIENTO	34
CAPÍTULO III	38
3. PROTOTIPO ELECTRÓNICO	38
3.1. DIAGRAMA DEL SISTEMA ELECTRÓNICO.....	38
3.2. PLATAFORMA DE DESARROLLO SPARK CORE	39
3.2.1. Micro – controlador STM32F103.....	39
3.2.2. Características de Spark Core	40
3.2.3. Configuración de conexión Wi-Fi	42
3.3. LECTOR RFID MFRC522.....	43
3.3.1. Interfaz de comunicación SPI	43
3.3.2. Registros y comandos	45
3.4. LIBRERÍA PARA MFRC522	46
3.5. CLIENTE TCP	49
3.6. DISEÑO DEL CIRCUITO IMPRESO O PCB.....	51
3.7. COSTOS	54
CAPÍTULO IV	55
4. APLICACIÓN WEB DEL SISTEMA.....	55
4.1. SERVIDOR <i>HTTP</i>	55
4.2. MySQL	56
4.3. PHP	57
4.4. XAMPP	58
4.5. DISEÑO Y CREACIÓN DE LA BASE DE DATOS	59
4.6. CakePHP.....	62
4.6.1. Arquitectura: Modelo Vista Controlador (MVC)	63
4.6.2. Ruteo.....	64
4.7. DISEÑO DE LA APLICACIÓN WEB	64
4.7.1. Ingreso (Login)	64
4.7.2. Dashboard de la aplicación.....	67
4.7.3. Administrar profesores, prototipos y aulas	67
4.8. INSTALACIÓN DE LA APLICACIÓN WEB	68



CAPÍTULO V	71
5. APLICACIÓN MÓVIL	71
5.1. INTRODUCCIÓN A ANDROID	71
5.1.1. Fundamentos de una aplicación.	72
5.1.2. Componentes de una aplicación.....	72
5.1.3. Recursos de una aplicación.....	73
5.2. ANDROID DEVELOPER TOOLS (ADT)	73
5.3. DESCRIPCIÓN GENERAL DE FUNCIONAMIENTO DE LA APLICACIÓN ..	74
5.4. DISEÑO E IMPLEMENTACIÓN.....	74
CONCLUSIONES Y RECOMENDACIONES	80
REFERENCIAS	82
ANEXO: CÓDIGOS DE PROGRAMACIÓN	84
A1.1. Programa del prototipo	84
A1.2. Aplicación Web	88
A1.3. API	94
A1.4. Aplicación Móvil	101

INDICE DE FIGURAS

Figura 1 Arquitectura Maestro - Esclavo de un sistema RFID. Fuente [1]	17
Figura 2 Componentes de un sistema RFID. Fuente [2]	17
Figura 3 Bus SPI Maestro - Esclavo. Fuente [4]	19
Figura 4 El modelo Cliente/Servidor en aplicaciones. Fuente [7]	21
Figura 5 Pila de protocolos TCP/Ip	22
Figura 6 Clientes en varios hosts conectándose al mismo puerto en un servidor. Fuente [18] ..	24
Figura 7 Múltiples clientes en un mismo host conectándose al mismo puerto en un servidor. Fuente [18]	24
Figura 8 Solicitud HTTP. Fuente [5].	25
Figura 9 Respuesta HTTP. Fuente [5].	26
Figura 10 Entidad en el modelo E - R	30
Figura 11 Relación uno a uno. Fuente [11]	30
Figura 12 Relación Uno a Muchos. Fuente [11]	31
Figura 13 Relación Muchos a Muchos. Fuente [11]	31
Figura 14 Elementos de una declaracion SQL. Fuente [13]	32
Figura 15 Tabla de una base de datos relacional. Fuente: El Autor	33
Figura 16 Diagrama del sistema propuesto. Fuente: El autor	34
Figura 17 Apertura mediante el uso de tarjeta. Fuente: El autor	36
Figura 18 Apertura mediante App. Fuente: El autor	37
Figura 19 Diagrama del prototipo electrónico. Fuente: El autor	38
Figura 20 Spark Core © Spark Team	39
Figura 21 STM32F103CBT6 en Spark Core v1.0 © Spark Team	40
Figura 22 Disposición de pines © Spark Team	42
Figura 23 Aplicación para configurar red WiFi en Spark Core © Spark Team	43
Figura 24 Conexión de RFID MFRC522 y Spark Core. Fuente: El autor	47
Figura 25 Variables para creación de cliente TCP. Fuente: El autor	49
Figura 26 Función para abrir una conexión al servidor. Fuente: El autor	50
Figura 27 Visualización de datos recibidos serialmente. Fuente: El autor	50
Figura 28 Código de Solicitud HTTP. Fuente: El autor	51
Figura 29 Circuito de control de cerradura. Fuente: El autor	52
Figura 30 Esquemático del PCB. Fuente: El Autor	52
Figura 31 Diseño de Circuito impreso o PCB. Fuente: El Autor	53
Figura 32 PCB implementado a doble capa. Fuente: El Autor	53
Figura 33 PCB implementado a doble capa con elementos posicionados y soldados. Fuente: El Autor	54
Figura 34 Código PHP embebido en código HTML. Fuente [22]	57
Figura 35 Página de Xampp instalado en Windows. Fuente: El autor	59
Figura 36 Panel de control de XAMPP. Fuente: El autor	59
Figura 37 Diagrama Entidad – Relación. Fuente: El Autor	60



Figura 38 Diagrama UML de la base de datos. Fuente: El Autor.....	60
Figura 39 Tabla profesores. Fuente: El autor	61
Figura 40 Tabla dispositivos. Fuente: El autor	61
Figura 41 Tabla aulas. Fuente: El autor	61
Figura 42 Tabla registros. Fuente: El autor	61
Figura 43 Página de login a la aplicación. Fuente: El autor	65
Figura 44 URL de login.....	65
Figura 45 Controlador de usuarios. Fuente: El autor	66
Figura 46 Vista de login. Fuente: El autor	67
Figura 47 Dashboard del administrador. Fuente: El autor.....	67
Figura 48Página para administrar profesores. Fuente: El autor	68
Figura 49 Formulario para agregar un nuevo profesor. Fuente: El autor	68
Figura 50 Configuración de la conexión de CakePHP con MySQL. Fuente: El autor	69
Figura 51 Configuración del archivo core.php. Fuente: El autor	70
Figura 52 Conexiones entre APP, API y API de Spark. Fuente: El autor	74
Figura 53 Pantallas de ingreso y selección de aula. Fuente: El autor	75
Figura 54 Solicitud Post para Login del usuario enviada desde Postman. Fuente: El autor	76
Figura 55 Respuesta JSON enviada desde el servidor a la App. Fuente: El autor	76
Figura 56 Listado de aulas en formato JSON, respuesta enviada desde el servidor. Fuente: El autor.....	77
Figura 57 Listado de aulas disponibles. Fuente: El autor	78
Figura 58 Respuesta enviada por la API de Spark. Fuente: El autor	79
Figura 59 Respuesta enviada desde el servidor a la App. Fuente: El autor	79



Universidad de Cuenca
Clausula de derechos de autor

Yo, *Christian Xavier Arias Ordóñez*, autor de la tesis "DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA PROTOTIPO DE GESTIÓN DE ACCESO A LAS AULAS DE LA FACULTAD DE INGENIERÍA MEDIANTE LLAVES ELECTRÓNICAS", reconozco y acepto el derecho de la Universidad de Cuenca, en base al Art. 5 literal c) de su Reglamento de Propiedad Intelectual, de publicar este trabajo por cualquier medio conocido o por conocer, al ser este requisito para la obtención de mi título de Ingeniero en Electrónica y Telecomunicaciones. El uso que la Universidad de Cuenca hiciere de este trabajo, no implicará afección alguna de mis derechos morales o patrimoniales como autor.

Cuenca, 17 de abril de 2015

Christian Xavier Arias Ordóñez

C.I: 0104741723



Universidad de Cuenca
Clausula de propiedad intelectual

Yo, Christian Xavier Arias Ordóñez, autor de la tesis "DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA PROTOTIPO DE GESTIÓN DE ACCESO A LAS AULAS DE LA FACULTAD DE INGENIERÍA MEDIANTE LLAVES ELECTRÓNICAS", certifico que todas las ideas, opiniones y contenidos expuestos en la presente investigación son de exclusiva responsabilidad de su autor.

Cuenca, 17 de abril de 2015

Christian Xavier Arias Ordóñez

C.I: 0104741723



DEDICATORIA

*A mi amiga, mi compañera para toda la vida,
mi esposa Gabby, por su amor incondicional,
por estar siempre allí, por cada sonrisa y
cada beso que me ayudó tantas veces a
levantarme después de cada tropiezo.*

*A mi princesa que se escapó de un cuento de
hadas, mi nenita Sammy, por la infinita
alegría e inspiración que me entregas, con
cada una de tus pequeñas locuras, haces que
la vida sea perfecta cada día.*

*A mi mami y mi papi, Martha y Washington,
éste triunfo de la vida no es solo mío,
también es suyo pues sin su apoyo e
incansable trabajo de seguro no lo hubiese
conseguido.*

Christian.



AGRADECIMIENTO

*Mi sincero agradecimiento al Ingeniero
Remigio Guevara, no solo por su acertada
guía para el desarrollo de ésta tesis,
también por su amistad y la confianza
depositada en mí durante toda mi época de
estudiante de la facultad.*

Christian



INTRODUCCIÓN

PRESENTACIÓN

El presente proyecto de tesis busca implementar una solución que permita a la Facultad de Ingeniería administrar de mejor manera el acceso a las aulas y laboratorios, automatizando la tarea que actualmente es realizada por personal de conserjería mediante un sistema prototipo electrónico e informático conectado a la intranet de la Universidad de Cuenca que permita a los docentes y alumnos acceder utilizando llaves electrónicas.

JUSTIFICACIÓN

Existe en la actualidad un auge de dispositivos inteligentes que combinan las capacidades de los campos de la electrónica con el de las telecomunicaciones y la informática. La automatización de procesos industriales se lo ha hecho hace varias décadas, a medida que la electrónica de estado sólido y los sistemas informáticos avanzaban, era posible disminuir el costo de este tipo de implementaciones. A medida que los dispositivos electrónicos se hacían accesibles al consumo del público en general, la automatización en el hogar empezó a ser un nuevo campo de desarrollo debido a que tareas cotidianas como encender o apagar un electrodoméstico o abrir la puerta del garaje han sido también automatizadas y requieren mínima o nula intervención del ser humano.

En la última década la humanidad ha presenciado un avance sin precedentes de la tecnología en todos los campos, la electrónica de estado sólido ha llegado a niveles de miniaturización impresionantes, lo que ha permitido disponer de dispositivos cada vez más poderosos en cuanto a velocidad de procesamiento y capacidad de almacenamiento se refiere. Además el internet es una tecnología a la que las personas tienen acceso cada vez en mayor número, convirtiéndose de a poco en un nuevo servicio básico como la energía eléctrica o el agua.

Estos hechos han permitido que la era del internet de las cosas haya comenzado pues en los últimos años han aparecido dispositivos inteligentes que pueden ser controlados y monitoreados por los usuarios desde cualquier



equipo que tenga una conexión a internet disponible como un computador, una Tablet o un teléfono inteligente.

Por estas razones y considerando que la Facultad de Ingeniería cuenta con una escuela enfocada en la Electrónica y las Telecomunicaciones creada hace menos de una década, se ha propuesto implementar una solución para el control de acceso a las aulas y laboratorios, de esta forma además de brindar mayor fluidez a las sesiones de clase eliminando la necesidad de la intervención del personal de conserjería al momento de abrir los espacios respectivos, se pretende dar además seguridad considerando que casi cada espacio en la facultad actualmente cuenta o contará con equipos multimedia en su interior.

Finalmente al plantear un prototipo conectado a internet la información generada en cuanto al ingreso de personal a los diferentes espacios podrá ser registrada y de ser necesario comparada con el distributivo de horas de cada aula o laboratorio.

En este proyecto de tesis se diseñará e implementará un dispositivo similar a los que están apareciendo en el mercado mundial pero con la ventaja de estar adaptado a las necesidades que se considera existen en la facultad, pues el sistema informático de administración de los dispositivos estará preparado para consumir información existente en otros sistemas de la universidad y además se garantiza la escalabilidad del sistema en caso de ser implementado en el futuro.

ALCANCE

En esta tesis se presenta información referente a la electrónica aplicada a la automatización de procesos mediante la conexión de dispositivos a internet que permite la interacción con servicios disponibles en la nube.

Conceptos relacionados con las características de los dispositivos seleccionados para la implementación del prototipo también son abordados, así como aquellos que están relacionados con la implementación de una aplicación y servicios web que permitan administrar el sistema completo.

El resultado del proyecto de tesis será un dispositivo electrónico prototipo que en conjunto con un sistema informático básicamente una aplicación y servicios web, podrá ser o no implementado en la Facultad de Ingeniería en el futuro como parte de otra tesis. Así también se deja la oportunidad de considerar el presente trabajo para ser mejorado u optimizado en futuros proyectos dentro de la carrera de Electrónica y Telecomunicaciones.



OBJETIVO GENERAL

Diseñar e implementar un dispositivo electrónico prototipo y un sistema informático que permita gestionar el acceso a las aulas y laboratorios de la Facultad de Ingeniería de la Universidad de Cuenca.

OBJETIVOS ESPECÍFICOS

- Diseñar e implementar un dispositivo electrónico prototipo capaz de leer tarjetas magnéticas, operar una cerradura y comunicarse con un sistema informático mediante una conexión inalámbrica a una red con acceso a internet.
- Implementar una aplicación web que permita registrar en la base de datos los dispositivos electrónicos instalados en cada aula o laboratorio, además de registrar a los docentes y sus respectivos códigos de acuerdo a la tarjeta magnética asignada.
- Desarrollar servicios web que permitan la comunicación del prototipo con un servidor *HTTP*, que permita el envío de los identificadores de cada docente hacia el servidor para su verificación.



CAPÍTULO I

1. MARCO TEÓRICO

1.1. INTRODUCCION

En este capítulo se abordan temas que se ha considerado son necesarios para una mejor comprensión de los siguientes capítulos por parte del lector. Cada uno de los temas se ha enfocado como una introducción a los mismos, de manera que los conceptos presentados justifiquen la aplicación de la tecnología o herramienta en la presente tesis y posean su respectivo sustento teórico. Si bien cada tema puede no estar relacionado con los otros de forma directa, la aplicación en conjunto permite cumplir los objetivos establecidos de la tesis.

1.2. RFID: IDENTIFICACION POR RADIO FRECUENCIA

Radio Frequency IDentification o RFID por sus siglas en el idioma inglés, es una tecnología electrónica de identificación cuyo funcionamiento está basado en el uso de campos electromagnéticos. A diferencia de sistemas más tradicionales de identificación como códigos de barras, RFID presenta ciertas ventajas que lo convierten en una opción a ser considerada, entre las cuales se encuentran la capacidad de almacenar mayor cantidad de información y que puede ser reprogramada. Idealmente las etiquetas o tarjetas con tecnología RFID no necesitan una fuente de energía o alimentación, ya que la misma proviene del lector cuando se realiza la transferencia de datos sin contacto.

1.2.1. Descripción y funcionamiento de un sistema RFID

Los sistemas RFID están compuestos generalmente por una aplicación o software que permite el funcionamiento del lector, además de una etiqueta. Se puede considerar que el sistema trabaja como una arquitectura Maestro – Esclavo, actuando el lector como Maestro y la etiqueta como esclavo, e inclusive si se considera a la aplicación como una parte independiente en el

sistema, actuaría como Maestro y el lector como Esclavo, esto se puede apreciar en la siguiente figura.

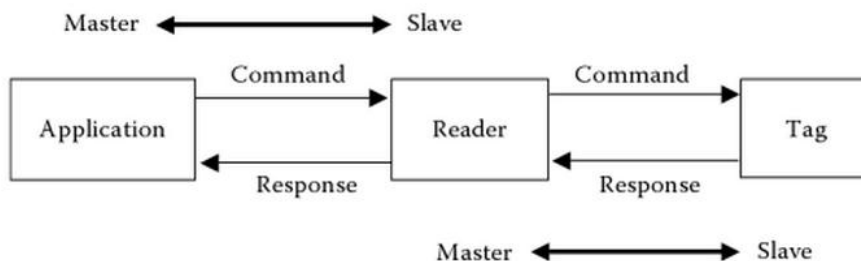


FIGURA 1 ARQUITECTURA MAESTRO - ESCLAVO DE UN SISTEMA RFID. FUENTE [1]

De forma general se describe al sistema el cual está formado dos partes principales, el lector y el transpondedor (etiqueta), considerando a la aplicación un complemento del sistema o como parte del lector. La aplicación puede ser un programa escrito para computadora o para un micro-controlador dependiendo de la aplicación que se requiera. En la siguiente figura se presenta un diagrama típico que representa al sistema RFID.

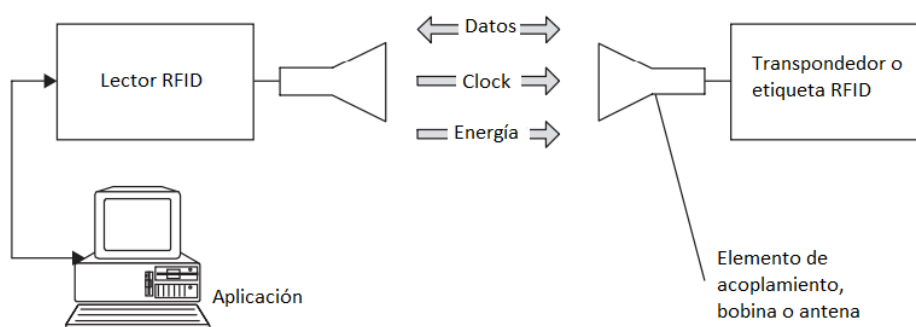


FIGURA 2 COMPONENTES DE UN SISTEMA RFID. FUENTE [2]

Funcionamiento.

El funcionamiento del sistema RFID es sencillo pues el lector posee una “zona de interrogación” que va desde unos pocos centímetros hasta varias decenas de metros, dependiendo del tipo de tecnología aplicada. El lector transmite una señal codificada que interroga a las etiquetas cercanas, una vez que el lector recibe la respuesta de una etiqueta, el intercambio de datos inicia. De esta manera se recibe el código de identificación único el cual será procesado por la aplicación o incluso si se ha grabado datos adicionales en la memoria no volátil de la etiqueta, se puede recibir otro tipo de información como número de lote, fecha de fabricación, etc.

El lector contiene un módulo de radio frecuencia el cual le permite transmitir y recibir datos, una unidad de control la cual varía dependiendo del fabricante y el estándar utilizado, además de un elemento de acoplamiento al transpondedor. Finalmente los lectores poseen varias interfaces de comunicación como *RS232*, *SPI*, *I2C* o *USB*, de esta manera es posible utilizar



estos dispositivos en la aplicación que se requiera, ya sea con un micro-controlador o una computadora.

Las etiquetas RFID internamente están formadas por lo menos de dos partes: Un circuito integrado el cual se encarga almacenar y procesar la información, modular y demodular la señal de radio frecuencia además de un elemento de acoplamiento que es básicamente una antena para recibir y transmitir los datos.

1.2.2. Etiquetas.

Existen básicamente dos tipos de etiquetas o *tags*:

Pasivas

No poseen fuente de alimentación, el campo magnético emitido desde el lector induce una corriente eléctrica mínima pero suficiente para que el circuito CMOS interno funcione.

Al no poseer una fuente de alimentación propia, la limitación de las tarjetas pasivas es la distancia de operación que alcanza unos pocos metros dependiendo de la frecuencia de operación y el tamaño de la antena.

Otra característica de las etiquetas pasivas es la necesidad de recibir una señal de gran intensidad por parte del lector, mientras que la respuesta emitida será una señal de muy baja intensidad.

Activas.

Este tipo de etiquetas posee internamente una fuente de alimentación, por lo general una batería cuya vida útil puede ser de varios años. La ventaja de este tipo de etiquetas es la capacidad de detectar y decodificar señales muy débiles provenientes de los lectores los cuales pueden estar ubicados a varias decenas de metros de distancia, y también al tener una fuente propia de energía pueden emitir señales de mayor intensidad que no se vean afectadas por la distancia o algún tipo de interferencia.

1.2.3. Frecuencias de operación.

Las frecuencias de operación de las etiquetas RFID, son las que se muestran en la siguiente tabla. Actualmente no existe una regulación mundial para el uso de las mismas, dejando a cada país la libertad de fijar las leyes necesarias en caso de que sea requerido.

Banda	Rango	Tasa de datos	Regulación	Costo aproximado de producción (US \$)
120 – 150 KHz (LF)	10 cm	Baja	Ninguna	\$ 1



13.56 MHz (HF)	10 cm – 1 m	Baja – Moderada	ISM band	\$ 0.50
433 MHz (UHF)	1 – 100 m	Moderada	Dispositivos de corto alcance	\$ 5
865 – 868 MHz (Europa) 902 – 928 MHz (EEUU) (UHF)	1 – 12 m	Moderada – Alta	ISM band	\$ 0.15 (pasivas)
2450 – 5800 MHz (Microonda)	1 – 2 m	Alta	ISM band	\$ 25 (activas)
3.1 – 10 GHz (Microonda)	200 m	Alta	Ultra wide band	\$ 5

TABLA 1 FRECUENCIAS DE OPERACIÓN. FUENTE [3]

1.3. INTERFAZ SPI

Anteriormente se mencionó que los lectores RFID poseen por lo general varias interfaces de comunicación entre las cuales se encuentra SPI o *Serial Peripheral Interface*. SPI es una interfaz usada comúnmente para transmitir datos entre micro-controladores y periféricos como sensores, memorias, registros de desplazamiento, etc. A continuación se presenta una breve descripción del funcionamiento de esta interfaz pues es la que se utiliza en el lector RFID seleccionado para la presente tesis. Se recomienda estar familiarizado con el funcionamiento de micro-controladores para mejor comprensión de este tema.

1.3.1. Funcionamiento

El bus SPI es un enlace de datos seriales síncronos, basado en la arquitectura Maestro – Esclavo, permite la comunicación entre un dispositivo maestro y varios dispositivos esclavo utilizando un número reducido de líneas o conexiones, como se indica en la siguiente figura.

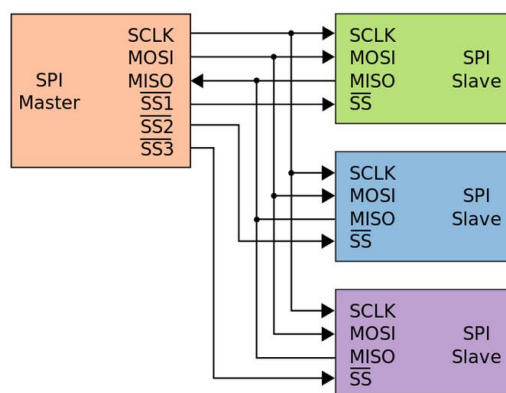


FIGURA 3 BUS SPI MAESTRO - ESCLAVO. FUENTE [4]



Se observa que el dispositivo Maestro, el cual puede ser un micro-controlador, comparte tres líneas (SCLK, MOSI, MISO) con los dispositivos Esclavos, mientras que existe solo una línea independiente (SS1, SS2,..., SSn) por Esclavo conectado.

El funcionamiento es simple, todas las líneas SS o *Slave Select* se mantienen en estado lógico alto. Cuando el Maestro necesita comunicarse con uno de los Esclavos, simplemente cambia la línea SS respectiva de alto a bajo, ésta es la señal (activo en bajo) que indica al dispositivo Esclavo que se ha iniciado la comunicación con el Maestro.

El Maestro coordina la transmisión de los datos pues se encarga del control de la señal SCLK o *Serial Clock*. Cuando los datos son enviados desde el Maestro al Esclavo, la línea MOSI *Master Out – Slave In* es utilizada, por el contrario si el Esclavo es quien transfiere datos entonces estos viajan por la línea MISO *Master In – Slave Out*.

Al ser el Maestro el único dispositivo encargado de la generación de pulsos de reloj, es necesario conocer de antemano cuando un dispositivo Esclavo necesita enviar datos de respuesta para que los pulsos se continúen generando hasta que toda la información sea recibida por el Maestro. Esto puede parecer un problema de comunicación, comparado con un sistema serial asíncrono en el cual la información puede ser enviada en cualquier dirección y en cualquier instante, pero en realidad no es así ya que por lo general, la comunicación SPI es utilizada para comunicarse con sensores u otros dispositivos que poseen un conjunto de comandos específicos los cuales siempre devuelven un número fijo de bytes; entonces es posible programar al micro-controlador para que de acuerdo al comando enviado al Esclavo, se prepare para recibir una cierta cantidad de bytes de respuesta.

1.4. ARQUITECTURA CLIENTE – SERVIDOR

Un *servidor* se lo define como una aplicación que ofrece un servicio a los usuarios de una red de computadoras, por el contrario un *cliente* es un programa que solicita un servicio. Una aplicación puede consistir de tanto un servidor como un cliente, los cuales pueden estar ejecutándose en el mismo o en diferentes sistemas.

Los usuarios usan la parte *cliente* en una aplicación, la cual se encarga de realizar las *solicitudes* para usar un servicio en particular, estas solicitudes son enviadas hacia el servidor usando *TCP/IP* como transporte, el cual será explicado en el siguiente punto.

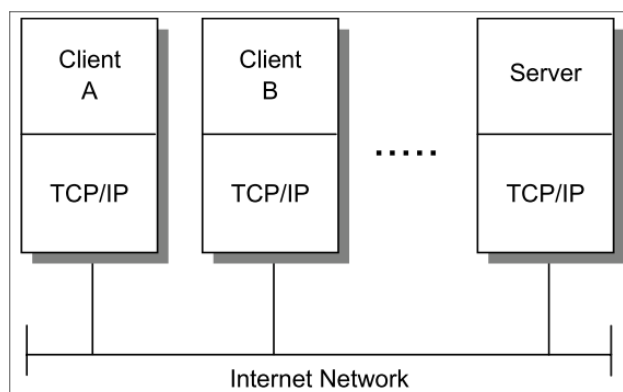


FIGURA 4 EL MODELO CLIENTE/SERVIDOR EN APLICACIONES. FUENTE [7]

El servidor es un programa encargado de recibir las solicitudes, realiza el servicio requerido y envía de regreso los resultados en una *respuesta*. Idealmente un servidor debe ser capaz de manejar varias solicitudes y varios clientes a la vez, por lo cual los programas servidores pueden clasificarse en dos tipos:

1. Iterativos: ya que iteran constantemente de la siguiente manera:
 1. Espera que llegue una solicitud de un cliente.
 2. Procesa la solicitud y envía una respuesta hacia el cliente.
 3. Regresa al paso 1.

De esta manera, los servidores iterativos pueden manejar varios clientes de forma secuencial, esto quiere decir que debe finalizar de procesar y responder la solicitud del cliente actual antes de servir al siguiente.

2. Concurrentes: funcionan de acuerdo a los siguientes pasos:
 1. Espera que llegue una solicitud de un cliente.
 2. Usa un nuevo *proceso/tarea/hilo* para atender la solicitud.
 3. Regresa al paso 1.

De esta manera es posible atender las solicitudes de varios clientes de manera paralela.

Los diseños iterativos son simples y su uso es adecuado para servicios de muy corta duración de manera que el tiempo de espera de ejecución por cliente sea mínimo. Por el contrario, aplicaciones servidores con diseño concurrente son más complejos pero poseen un rendimiento superior ya que se mejora los tiempos de respuesta y reduce la latencia es decir la tasa de procesamiento de solicitudes es menor que la tasa de llegada de las mismas. Ejemplos de servicios concurrentes son **HTTP**, **Telnet** o **FTP**.



1.5. MODELO TCP/IP

Como se mencionó en el tema anterior, el transporte de mensajes en una arquitectura o modelo *Cliente – Servidor* se lo realiza sobre *TCP/IP* por lo cual es importante explicar brevemente su funcionamiento.

El nombre del modelo **TCP/IP** se debe a los dos protocolos más importantes que lo conforman: **Transmission Control Protocol** e **Internet Protocol**.

TCP/IP está modelado en capas, esta representación hace que al modelo también se lo conozca como una pila o *stack* de protocolos. Al dividir el software de comunicación en capas, la pila de protocolos permite la división de tareas y por lo tanto facilita la implementación. Las capas se comunican ya sea con sus capas superiores o inferiores a través de las interfaces respectivas, *de esta manera cada capa provee servicios directamente a su capa superior y hace uso de los servicios de la capa inferior*. Por ejemplo la capa IP provee la habilidad de transferir datos entre hosts pero no garantiza la entrega de los mismos, el protocolo de transporte *TCP* hace uso de dicho servicio para proveer aplicaciones con entrega de paquetes de datos garantizando su confiabilidad y el orden adecuado. La siguiente figura ilustra el modelo TCP/IP.

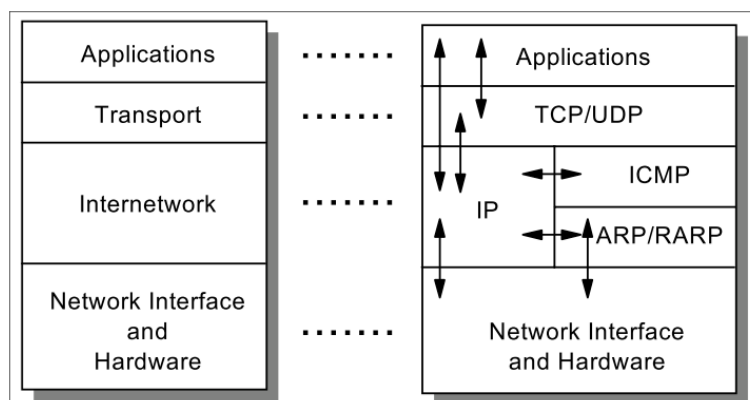


FIGURA 5 PILA DE PROTOCOLOS TCP/IP

1.5.1. Aplicaciones TCP/IP

Los protocolos de más alto nivel dentro del modelo, son aquellos correspondientes a la capa de aplicación, los cuales permiten la comunicación entre aplicaciones en hosts de redes diferentes, estas aplicaciones son las únicas interfaces visibles del modelo TCP/IP para los usuarios.

Todas las aplicaciones tienen algunas características en común:



- Pueden ser aplicaciones desarrolladas en cualquier lenguaje por los usuarios o aplicaciones estándar *Telnet, FTP, SMTP*, etc.
- Estas aplicaciones utilizan ya sea *UDP* o *TCP* como protocolo de transporte. *UDP* es un protocolo no orientado a la conexión, lo cual quiere decir que un host realiza el envío de paquetes a otro host sin que éste último pueda confirmar la recepción, este protocolo emplea un flujo de datos unidireccional.
TCP por el contrario es orientado a la conexión, lo cual sin entrar en detalles permite control de errores, recepción en orden de los paquetes de datos y confirmación de recepción debido a que el protocolo es bidireccional, en caso de pérdida o errores de un paquete el host receptor puede solicitar al emisor que vuelva a enviar los datos.
- La mayoría de aplicaciones usan el modelo Cliente – Servidor.

1.5.2. Puertos y Sockets

Las aplicaciones Cliente – Servidor se comunican mediante el envío de flujos de bytes o datos mediante una conexión, por lo general usando el protocolo de transmisión *TCP*. Cada proceso que se desea comunicar con otro, se identifica a sí mismo con la pila *TCP/IP* mediante uno o más puertos.

Un **puerto** como se define en [7, pg. 144] es un número de 16 bits usado para identificar a que protocolo o aplicación de alto nivel se debe entregar los mensaje o flujo de bytes que llegan al receptor.

Existen dos tipos de puertos:

- **Puertos de sistema** (*well-known*): son puertos definidos desde el número 0 al 1023, se los conoce como puertos de sistema debido a que pertenecen a servicios estándar, como por ejemplo: *Telnet* (23), *FTP* (20, 21), *HTTP* (80), *HTTPS* (443), por nombrar algunos.
- **Efímeros o asignados dinámicamente**: son aquellos usados por programas que no usan puertos definidos de sistema debido a que la comunicación la realizan a través de los puertos disponibles desde el número 1024 al 65535, y son usados solo el tiempo necesario para cada aplicación en particular.

La combinación de una dirección IP y un puerto, es llamado **socket**. Un socket identifica de forma única a un proceso de red dentro de todo Internet. Un par de sockets, uno para el host que recibe y otro para el host que envía, definen la conexión para protocolos orientados a la conexión como *TCP*.

Técnicamente se definen los siguientes términos cuando se refiere a sockets:

- *Socket address*, es una tripleta de la forma:
<protocol, dirección local, puerto local>

Por ejemplo:

<tcp, 192.168.14.223, 8080>

- Una *conversación* es el enlace de comunicación entre dos procesos.
- Una *asociación* es una 5 – tupla que especifica completamente los dos proceso que intervienen en una conexión:
<protocol, dirección local, puerto local, dirección foránea, puerto foráneo>

Por ejemplo:

<tcp, 192.168.14.223, 1500, 192.168.44.12, 22>

Cuando dos procesos que se comunican a través de sockets TCP, el modelo de sockets provee conexión full – dúplex entre los dos. Las siguientes figuras ilustran la idea de conexión mediante *sockets*:

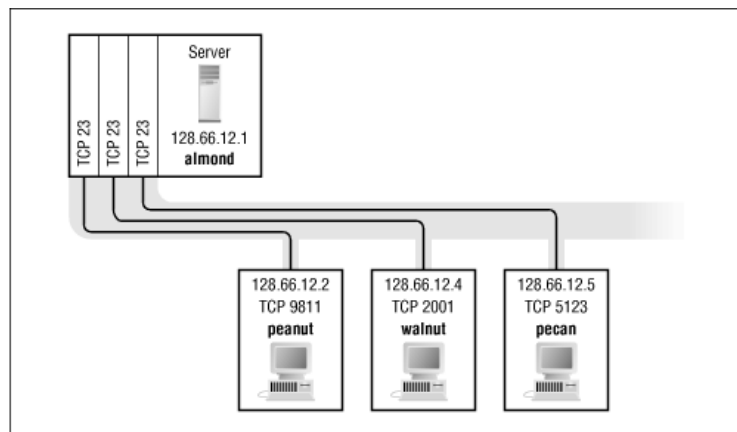


FIGURA 6 CLIENTES EN VARIOS HOSTS CONECTÁNDOSE AL MISMO PUERTO EN UN SERVIDOR. FUENTE [18]

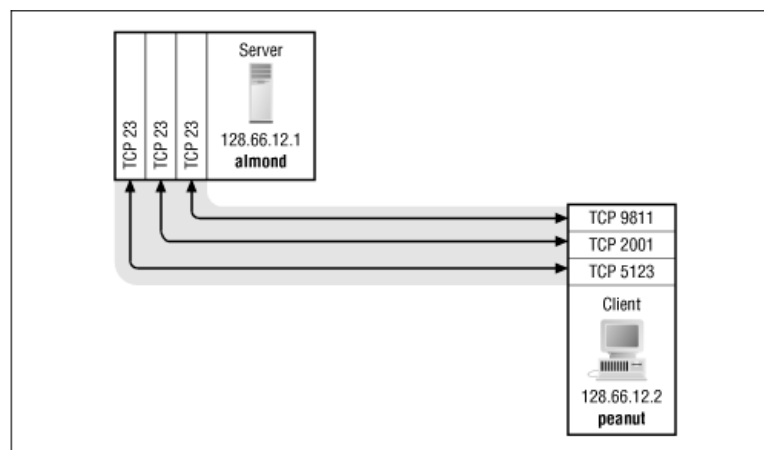


FIGURA 7 MÚLTIPLES CLIENTES EN UN MISMO HOST CONECTÁNDOSE AL MISMO PUERTO EN UN SERVIDOR. FUENTE [18]



1.6. PROTOCOLO HTTP

HTTP (*HyperText Transfer Protocol*) es un protocolo perteneciente a la capa de aplicación del modelo de referencia TCP/IP, que básicamente permite la transmisión de datos (hipertexto) mediante el protocolo TCP. Es un protocolo simple que fue originalmente desarrollado para obtener recursos estáticos basados en texto.

HTTP usa mensajes, una interacción solicitud – respuesta es realizada, lo cual referencia al protocolo con el conocido modelo computacional *cliente – servidor*, de esta manera es sencillo comprender que un cliente realiza una solicitud al servidor, éste se encarga de recibir el mensaje y responder con el recurso solicitado por el cliente.

Aunque ya se mencionó que el protocolo usa TCP como protocolo de transporte, HTTP es un protocolo *no orientado a la conexión*, lo cual puede sonar contradictorio conociendo que la función del protocolo TCP es garantizar que los datos sean entregados a su destino sin errores y en el orden en el que fueron transmitidos, pero esto se puede comprender al considerar que cada interacción *solicitud – respuesta* es autónoma de la anterior o la siguiente ya que cada una utiliza una conexión TCP distinta, de esta manera en caso de ser necesario retransmitir un recurso el canal de comunicación será una nueva conexión TCP.

1.6.1. Solicitudes HTTP

Las solicitudes y respuestas HTTP consisten de una o más cabeceras, cada una en una línea diferente, seguida por un espacio en blanco obligatorio, seguido finalmente por un cuerpo de mensaje opcional. Una solicitud típica es la que se muestra en la siguiente figura:

```
GET /auth/488/YourDetails.ashx?uid=129 HTTP/1.1
Accept: application/x-ms-application, image/jpeg, application/xaml+xml,
image/gif, image/pjpeg, application/x-ms-xbap, application/x-shockwave-
flash, */*
Referer: https://mdsec.net/auth/488/Home.ashx
Accept-Language: en-GB
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64;
Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR
3.0.30729; .NET4.0C; InfoPath.3; .NET4.0E; FDM; .NET CLR 1.1.4322)
Accept-Encoding: gzip, deflate
Host: mdsec.net
Connection: Keep-Alive
Cookie: SessionId=5B70C71F3FD4968935CDB6682E545476
```

FIGURA 8 SOLICITUD HTTP. FUENTE [5].

La primera línea de cada solicitud HTTP consiste de tres ítems, separados por espacios:



- Una palabra (verbo) que indica el método HTTP. El método más usado es GET, cuya función es solicitar o recuperar recursos disponibles en un servidor web. GET no posee un cuerpo de mensaje, así que no hay datos después de la línea en blanco que sigue a las cabeceras. Los verbos disponibles son los que se indican a continuación:

GET	Obtiene un recurso del servidor
HEAD	Obtiene solo las cabeceras relacionadas a algún recurso disponible en el servidor.
POST	Envía datos al servidor para que sean procesados, éste método permite cabeceras y cuerpo de mensaje.
PUT	Envía un recurso al servidor, para hacerlo requiere que el mensaje contenga cuerpo.
OPTIONS	Averigua los verbos que admite un servidor específico.
DELETE	Elimina un recurso del servidor.

- La solicitud URL. La URL funciona como un nombre para el recurso solicitado, junto con una cadena de consulta opcional que contiene parámetros que el cliente pasa al recurso. La cadena de consulta se encuentra después del carácter ? en la URL. En la imagen se puede observar que se pasa el parámetro *UID* cuyo valor es 129.
- La versión HTTP usada que por lo general es la 1.1.

1.6.2. Respuestas HTTP

La respuesta enviada por el servidor después de procesar la solicitud, se compone de un número que indica el resultado de la petición, cabeceras sobre el recurso y finalmente el recurso en sí mismo. Una respuesta típica es como sigue:

```
HTTP/1.1 200 OK
Date: Tue, 19 Apr 2011 09:23:32 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Set-Cookie: tracking=ti8rk7joMx44S2Uu85nSWc
X-AspNet-Version: 2.0.50727
Cache-Control: no-cache
Pragma: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 1067

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"><html xmlns="http://
www.w3.org/1999/xhtml" ><head><title>Your details</title>
...
```

FIGURA 9 RESPUESTA HTTP. FUENTE [5].



La primera línea de cada respuesta HTTP consiste de tres ítems, separados por espacios:

- La versión HTTP usada.
- Un código numérico que indica el resultado de la solicitud. 200 es el código de estado más común, significa que la solicitud fue exitosa y que el recurso solicitado está siendo devuelto.
- Una leyenda textual que ayuda a interpretar el significado del número. OK ya que 200 quiere decir que el servidor está devolviendo o enviando el recurso solicitado pues no hubo error. Los posibles códigos son los siguientes:

1xx	Información
2xx	Operación exitosa
3xx	Redirección
4xx	Error del lado del cliente (petición)
5xx	Error del lado del servidor

1.6.3. URL

Como ya se mencionó anteriormente una URL se puede considerar un nombre para el recurso solicitud, las siglas significan *Uniform Resource Locator* y técnicamente es un identificador único para un recurso web.

El formato de la mayoría de las URLs es como sigue:

`protocol://hostname[:port]/[path/]file[?param=value]`

No todos los componentes son necesarios. El número del puerto es solo incluido en casos en los que difiere del predeterminado por el protocolo.

1.7. SISTEMA GESTOR DE BASE DE DATOS

Un Sistema Gestor de Base de Datos (SGBD) o *Data Base Management System (DBMS)*, “es una colección de datos relacionados entre sí, estructurados y organizados, y un conjunto de programas que acceden y gestionan esos datos. La colección de esos datos se denominan **Base de Datos (BD)** o **Data Base (DB)**.” [10].

Los SGBD están diseñados para permitir almacenar o extraer, grandes cantidades o bloques de datos, de manera eficiente y segura debido a la estructuración y mecanismos utilizados para gestionar dicha información.

Los SGBD permiten a los usuarios definir, crear y mantener una base de datos, ya sea mediante el uso de programas (clientes) propios o desarrollados por terceros, en ambos casos el SGBD debe proporcionar un acceso controlado a la información disponible.



Los Sistemas Gestores de Base de Datos, independientemente del desarrollador de los mismos, deben prestar servicios como:

- Creación y definición de una Base de Datos: se refiere al hecho de especificar el tipo de estructura y de datos, las relaciones y restricciones existentes entre los datos almacenados.
- Manipulación de datos: permitiendo al usuario realizar *consultas*, insertar, actualizar o eliminar datos.
- Control de acceso a los datos almacenados: mediante mecanismos o políticas de seguridad para permitir o restringir el acceso a los datos por parte de los usuarios.
- Concurrencia: mediante ésta característica es posible que varios usuarios accedan a los datos a la vez y puedan manipular los mismos.
- Mantener la integridad y consistencia de los datos, además de contar con mecanismos que permitan respaldar, exportar y recuperar información.

1.7.1. Componentes de los SGBD

Los Sistemas Gestores de Base de Datos, son programas muy complejos debido a que deben permitir manejar de forma eficiente grandes cantidades de datos, sus componentes principales son los siguientes:

- Lenguaje.

Existen varios lenguajes utilizados en diferentes SGBD, los cuales permiten al administrador especificar los datos que componen la Base de datos, la estructura y relaciones que existen en la misma, las reglas y controles de acceso.

- Diccionario de datos.

Es el lugar en donde se deposita la información acerca de los datos que forman la Base de Datos. *“El diccionario contiene las características lógicas de los sitios donde se almacenan los datos del sistema, incluyendo nombre, descripción, alias, contenido y organización.”* [10].

- Mecanismos de seguridad e integridad de datos.
- Administrador de la Base de Datos.

1.7.2. Modelos de datos

La descripción de una BD mediante un modelo de datos es conocido como **esquema de la base de datos**. Los modelos existentes son clasificados en modelos lógicos basados en objetos, modelos lógicos basados en registros y modelos físicos de datos.

- Modelos lógicos basados en objetos.



Los modelos más conocidos son:

- Entidad – Relación (E –R)
- Orientados a objetos

La mayoría de bases de datos relacionales añaden extensiones para poder ser relacionales – orientadas a objetos.

- Modelos lógicos basados en registros.

Los modelos de datos de este tipo más conocidos son:

- Modelo relacional
- Modelo de red
- Modelo jerárquico

El **modelo relacional** es muy importante ya que la mayoría de Gestores de Bases de Datos son relacionales, como en el caso del SGBD que se utiliza en la presente tesis, MySQL. Este modelo representa los datos y las relaciones entre los datos mediante tablas, cada una de las tablas posee columnas con nombres únicos que representan los atributos y filas que representan cada registro, las filas también son llamadas tuplas.

1.7.3. Modelo Entidad – Relación

Es un modelo potente para representar datos, haciendo uso de grafos y tablas. Para hacer uso de este modelo es necesario conocer ciertos conceptos básicos, los cuales son usados en el mismo:

- **Entidad:** es un objeto que existe en el mundo real, por ejemplo PERSONAS, CLIENTES, FACTURAS, etc.
- **Entidad fuerte:** es una entidad la cual no requiere de la existencia de otra para su propia existencia, por ejemplo PERSONAS, en cambio la entidad FACTURAS depende de la existencia de la entidad PERSONAS para su existencia
- **Atributos:** Son propiedades que describen a una entidad. Por ejemplo una PERSONA posee un número de cédula, nombre, apellido, dirección, estado civil, etc. Estas propiedades o características son los atributos que describen a una persona.
- **Identificador:** es el conjunto de atributos que pueden identificar de manera única a cada entidad, por ejemplo en la entidad EMPLEADOS, a cada empleado se lo puede identificar por su número de cédula, seguro social, etc.



- **Clave primaria o Primary Key:** es la clave seleccionada por el diseñador de la Base de Datos para identificar de manera única.
- **Clave foránea o foreign key:** es el atributo o conjunto de atributos de una entidad que forman la clave primaria en otra entidad.
- **Relaciones:** Se define como relación a la asociación entre diferentes entidades, las relaciones usan verbos como su nombre que permite identificar de otras relaciones y se representa mediante un rombo en los diagramas.

Diagramas de estructuras de datos en el modelo Entidad - Relación.

Permiten la representación de manera gráfica mediante símbolos, de la estructura de una base de datos. Los símbolos utilizados son los siguientes:

- Rectángulos para representar las entidades.
- Elipses para los atributos. Las claves primarias deben ser subrayadas.
- Rombos para las relaciones.

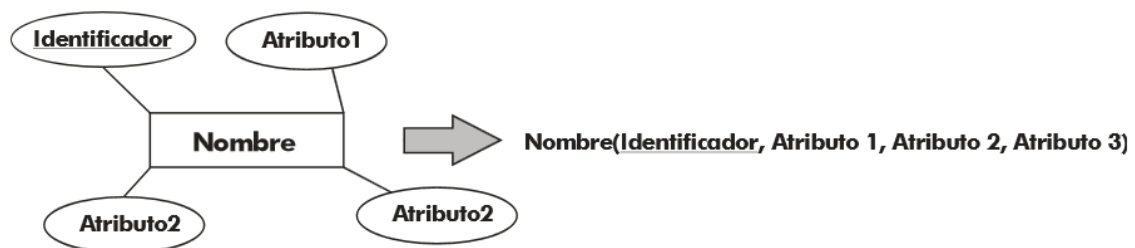


FIGURA 10 ENTIDAD EN EL MODELO E - R

Se considera en los diagramas E – R la cardinalidad de la relación que se describe, por ejemplo una PACIENTE tiene una HISTORIA_CLINICA o una PERSONA tiene varias FACTURAS.

- **1:1, uno a uno:** “A cada elemento de la primera entidad le corresponde sólo uno de la segunda entidad, y a la inversa” [31].



FIGURA 11 RELACIÓN UNO A UNO. FUENTE [11]

- **1:N, uno a muchos:** “A cada elemento de la primera entidad le corresponde uno o varios elementos de la segunda entidad, y a cada elemento de la segunda entidad le corresponde un solo elemento de la primera” [31].

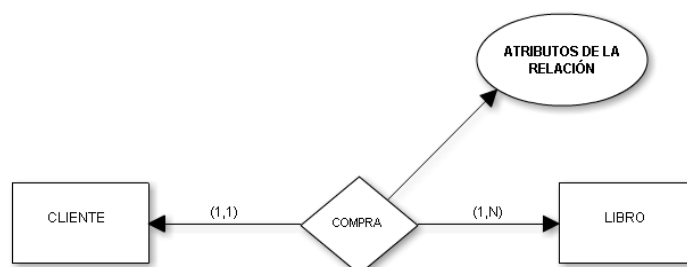


FIGURA 12 RELACIÓN UNO A MUCHOS. FUENTE [11]

- **M:N, muchos a muchos:** “A cada elemento de la primera entidad le corresponde uno o varios elementos de la segunda entidad, y a cada elemento de la segunda entidad le corresponde uno o más elementos de la primera entidad” [31].

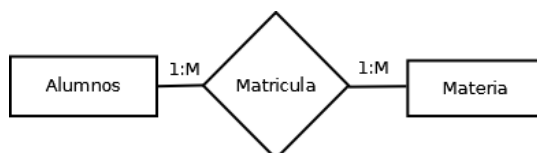


FIGURA 13 RELACIÓN MUCHOS A MUCHOS. FUENTE [11]

1.8. LENGUAJE SQL

SQL (*Structured Query Language*) es un lenguaje de programación ampliamente usado para trabajar con bases de datos relacionales, que como ya se mencionó anteriormente son bases de datos que implementan el almacenamiento de datos en base a las reglas de diseño del modelo relacional.

Los *tokens* y la sintaxis de SQL están orientados al lenguaje Inglés para mantener la barrera de acceso lo más pequeña posible, como por ejemplo de un comando escrito en SQL, se tiene:

```
SELECT `name` FROM `teachers` WHERE `cedula`=0104741723;
```

No es complicado deducir que este comando se usa para seleccionar el nombre del profesor que posee dicho número de cedula. Aunque ciertas palabras reservadas para el lenguaje en este caso fueron escritas en mayúsculas, SQL permite combinar ya sea mayúsculas o minúsculas solo para las *key words* sin ningún problema, por ejemplo:

```
Select, SELECT, SElect, SelEcT.
```

El núcleo del lenguaje SQL consiste de sentencias, declaraciones o *statements*, los cuales consisten de palabras claves o reservadas, operadores, valores, nombres del sistema y funciones. Como se puede observar en la sentencia de ejemplo, siempre los *statements* deben finalizar con un punto y coma.



1.8.1. Elementos del lenguaje

Las declaraciones SQL empiezan con una palabra clave como SELECT, DELETE o CREATE, y finalizan con un punto y coma (;). En la siguiente imagen se puede observar los elementos de una sentencia SQL.

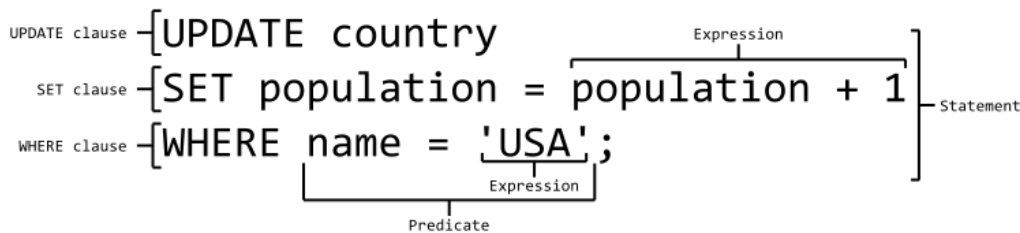


FIGURA 14 ELEMENTOS DE UNA DECLARACION SQL. FUENTE [13]

- *Clauses*: las declaraciones SQL están subdivididas en clausulas.
- *Predicates*: especifican condiciones las cuales son evaluadas y retornan valores booleanos, falso o verdadero, 1 ó 0. Por ejemplo para comparar se puede usar BETWEEN, LIKE, IS NULL, etc.
- *Expressions*: Son expresiones numéricas o cadenas de caracteres (Strings), aunque también pueden ser el resultado de operadores aritméticos o de unión, así como funciones.
- *Object names*: nombres de objetos de la base de datos como tablas, vistas, columnas, funciones.
- *Valores*: numéricos o cadenas de caracteres.
- *Operadores aritméticos*
- *Operadores de concatenación*.
- *Operadores de comparación*: igual (=), no es igual (<>), mayor que (>), menor que (<), mayor o igual que (>=), menor o igual que (<=).
- *Operadores booleanos*: AND, OR, NOT.

1.9. BASE DE DATOS RELACIONAL

Anteriormente ya se mencionó el modelo relacional, el cual es el más usado dentro de los sistemas de bases de datos, por esta razón es importante considerar un espacio para describir las bases de datos relacionales.

Una base de datos relacional está basada en el uso de tablas, las cuales para los usuarios son representadas con figuras rectangulares formadas por filas y columnas. Cada columna posee un nombre único y cada fila asigna un valor a cada columna formando una tupla o registro, por ejemplo:



cedula	name	last_name	uid
0104741723	Christian	Arias	9E6F17C
0106551419	Gabriela	Bulgarin	6E634398

FIGURA 15 TABLA DE UNA BASE DE DATOS RELACIONAL. FUENTE: EL AUTOR

1.9.1. Claves

Son atributos que identifican unívocamente a cada tupla de la tabla, como por ejemplo un número de cédula, el cual es único para cada persona y permite identificarla.

Las claves pueden ser: candidatas, primaria, alternativa, externa o foránea como ya se explicó en el modelo entidad – relación.

1.9.2. Restricciones

Las restricciones son condiciones que deben ser cumplidas de forma obligatoria por los datos. Las restricciones donde dos tipos:

- **Inherentes:** Son aquellas propias de las bases de datos relacionales, como por ejemplo el hecho de que por la misma estructura no pueden existir dos tuplas iguales en una misma tabla.
- **Semánticas:** Son aquellas que los usuarios incorporan o establecen para los datos como por ejemplo decidir el atributo que será clave primaria y que por lo tanto debe ser único para cada tupla. La elección de la clave primaria es una restricción semántica pues es el diseñador de la base de datos el que la establece.

Otras restricciones semánticas son:

- **Unicidad:** los atributos no son claves primarias pero tampoco pueden repetirse en las tuplas.
- **Obligatoriedad:** el atributo debe tener siempre un valor, no puede ser *nulo*.
- **Regla de validación:** es la condición o condiciones que un dato en concreto debe cumplir, por ejemplo para un identificador numérico no puede el valor contener letras u otros caracteres.

CAPÍTULO II

2. DISEÑO DEL SISTEMA

Este capítulo presenta el diseño que se consideró adecuado para desarrollar el presente proyecto de tesis. Inicialmente se presenta un diagrama que incluye todos los componentes que formarán parte del sistema para luego explicar brevemente su objetivo y funcionamiento debido a que los mismos serán detallados en los siguientes capítulos.

2.1. DIAGRAMA DEL SISTEMA

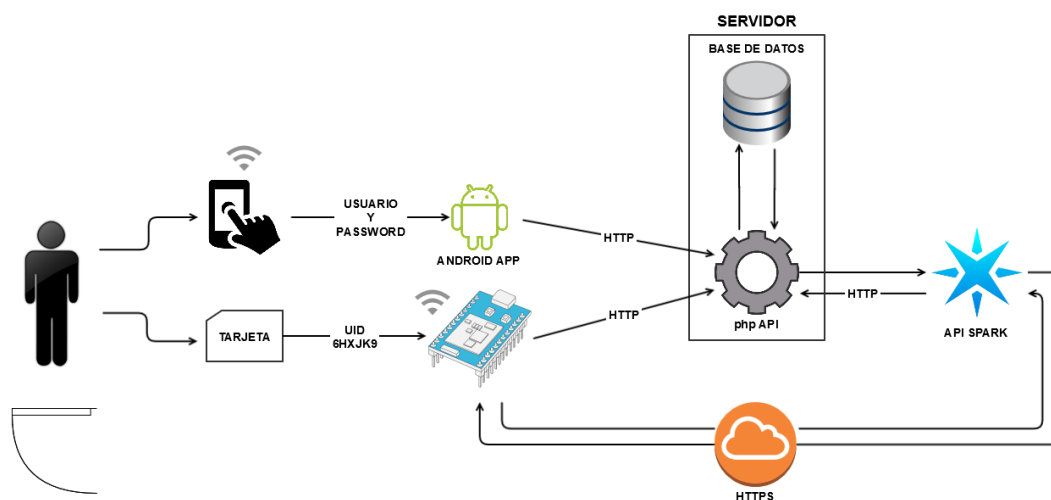


FIGURA 16 DIAGRAMA DEL SISTEMA PROPUESTO. FUENTE: EL AUTOR

2.2. DESCRIPCIÓN Y FUNCIONAMIENTO

El sistema propuesto está compuesto por un prototipo electrónico, el cual consta de una placa de desarrollo Spark Core que integra un módulo de conexión Wi-Fi para acceso a Internet. Acompañando a la placa de desarrollo, se encuentra un lector de tarjetas RFID el cual lee etiquetas (tags) o tarjetas creadas bajo el estándar ISO/IEC 14443 como las MIFARE de la compañía NXP Semiconductors, las cuales son comunes a los usuarios pues en el medio



ya han sido implementadas en el sistema de transporte público como forma de pago electrónico, esto en la ciudad de Cuenca.

La función del prototipo y el lector RFID es simple, recibir el código o UID de la tarjeta y mediante la conexión a internet, enviarla a un servidor para ser procesada. El servidor en mención es un Apache HTTP Server en conjunto con MySQL como Sistema Gestor de Base de datos y un intérprete de lenguaje PHP, éste servidor puede ser instalado en cualquier PC de escritorio o portátil.

Además de usar la tarjeta como llave electrónica, se ha creado una aplicación móvil para el sistema operativo Android con el mismo objetivo. Cada profesor tendrá asignado un nombre de usuario y contraseña, estas credenciales permiten acceder a la aplicación la cual para verificar las mismas se conecta al servidor mediante una API, entonces el usuario deberá seleccionar el aula o laboratorio que desea abrir.

La API es un conjunto de funciones programadas en lenguaje PHP a las cuales se puede acceder a través del protocolo HTTP y permiten la comunicación entre los componentes de software que integran el sistema. Para el proceso de verificación del código de la tarjeta de usuario, desde el Spark Core se establece una conexión (Cliente Socket TCP) con el servidor y se hace uso de una función de la API la cual recibe el UID, verifica el mismo en la base de datos y registra el acceso del usuario para luego haciendo uso de la API de Spark, activar el circuito que abre la puerta del aula o laboratorio mediante una función programada en la placa de desarrollo, la cual puede ser ejecutada mediante la conexión segura HTTPS que existe entre el servidor de Spark y cada Core.

Todo Spark Core al conectarse a la red WiFi, establece una conexión HTTPS con los servidores de Spark, lo cual permite mediante el uso del identificador único de cada dispositivo y un token de acceso, ejecutar funciones o recibir datos desde el dispositivo.

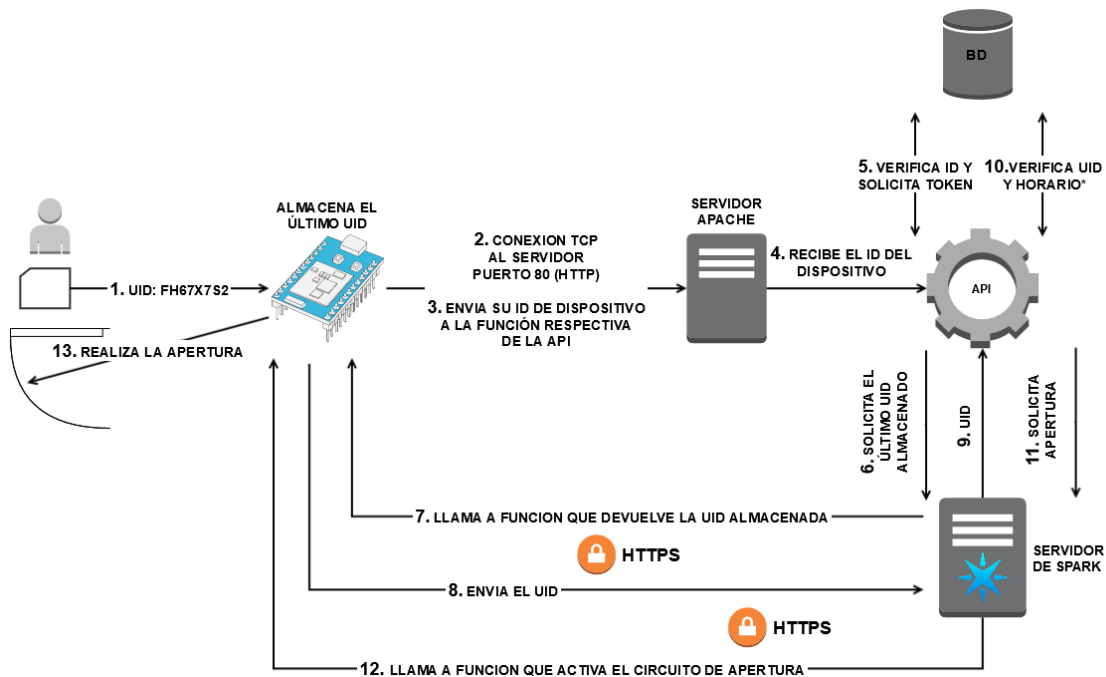


FIGURA 17 APERTURA MEDIANTE EL USO DE TARJETA. FUENTE: EL AUTOR

De igual manera al usar la aplicación móvil y después de autenticar al usuario mediante la API desarrollada en PHP, la apertura de la puerta se realiza de la misma manera, desde nuestro servidor se ejecuta la función que activa el circuito de apertura a través de la API de Spark y su conexión HTTPS con cada Core.

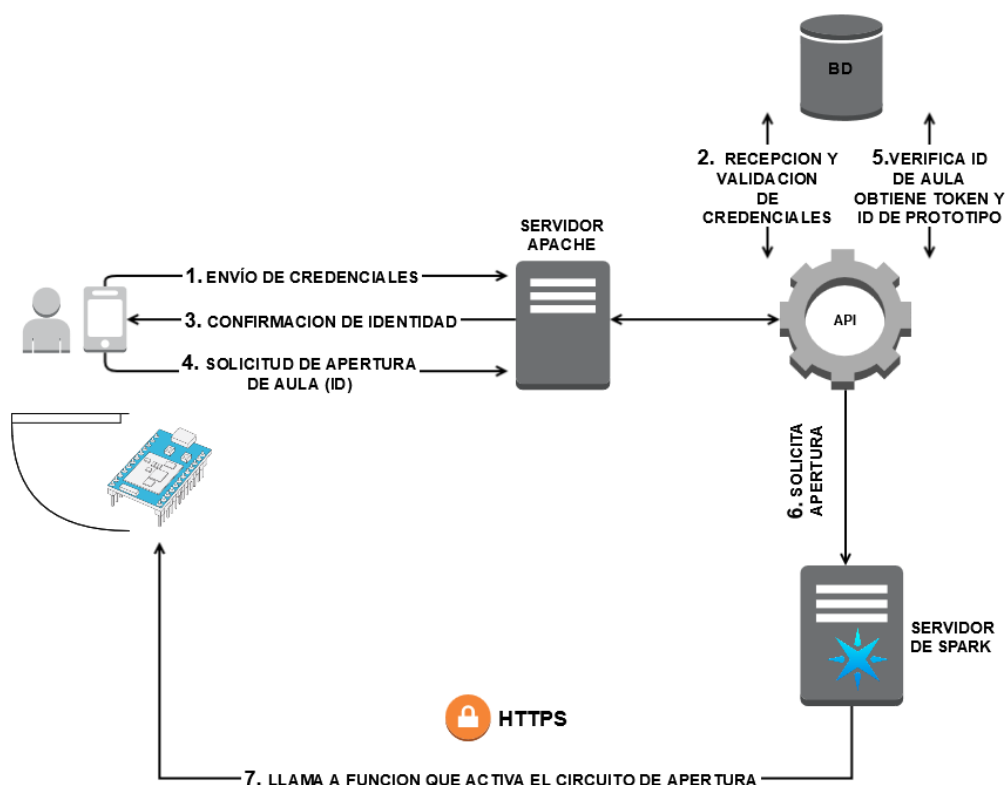


FIGURA 18 APERTURA MEDIANTE APP. FUENTE: EL AUTOR

Adicionalmente y como parte administrativa del sistema, se ha desarrollado una aplicación web la cual será instalada en el servidor Apache y podrá ser utilizada desde cualquier dispositivo con una conexión a Internet dentro de la Facultad de Ingeniería. El objetivo de la misma es administrar la información de la base de datos del sistema que incluye ingresar, modificar o eliminar profesores, aulas y dispositivos de acuerdo al diseño de la BD y la relación existente entre sus datos.

En los gráficos anteriores, algunos pasos han sido omitidos priorizando resaltar los más importantes. De esta manera se describe de forma breve el funcionamiento del sistema, cada una de sus partes, prototipo electrónico, servidor, aplicación web y aplicación móvil, serán detalladas en los siguientes capítulos.

CAPÍTULO III

3. PROTOTIPO ELECTRÓNICO

En este capítulo se considera el diseño del prototipo electrónico, los componentes que lo integran son presentados teniendo en cuenta que la información entregada en los anteriores capítulos fueron solo como referencia y no especificaba ningún dispositivo electrónico, casa fabricante o modelo de comercial.

El sistema electrónico lo integra un micro-controlador encargado de gestionar la comunicación inalámbrica mediante un módulo Wi-Fi, además el micro-controlador mediante una interfaz SPI se comunica con un lector RFID que trabaja en la frecuencia de 13.56 MHz cuyas prestaciones son adecuadas para el desarrollo de la presente tesis.

3.1. DIAGRAMA DEL SISTEMA ELECTRÓNICO

En la siguiente imagen se puede observar un diagrama general de los elementos que componen el sistema electrónico. Cada una de las partes que se indican a continuación, serán detalladas más adelante en el presente capítulo.

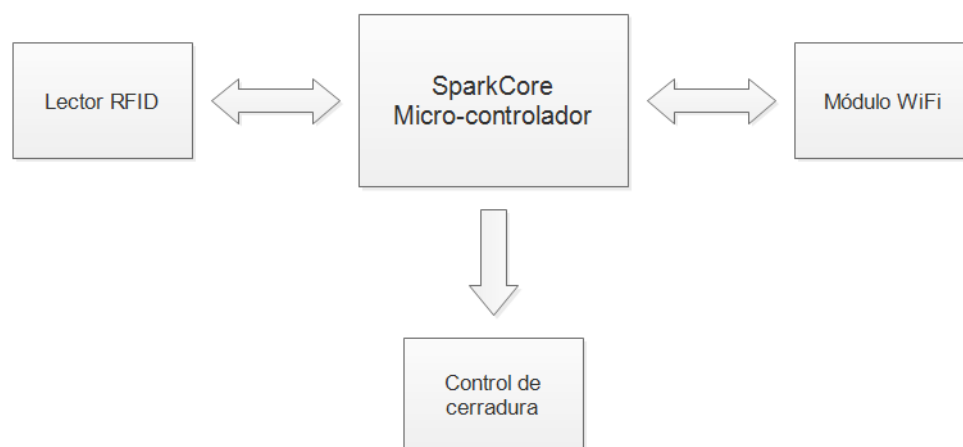


FIGURA 19 DIAGRAMA DEL PROTOTIPO ELECTRÓNICO. FUENTE: EL AUTOR

Como se puede observar en la figura, el micro-controlador es el encargado de la gestión de los otros elementos que componen el sistema electrónico. La forma de las flechas en el diagrama indican el sentido de la transferencia de datos, en el caso por ejemplo del lector RFID, es necesario establecer una “conversación” ya que como se detallará más adelante, el lector seleccionado tiene también un micro-controlador el cual recibe y procesa los diferentes comandos. De igual manera sucede con el módulo Wi-Fi, mientras que con la parte de control de la cerradura no existe una comunicación bidireccional ya que es el micro-controlador el encargado de activar o desactivar el circuito del mecanismo que permite el control de la misma.

3.2. PLATAFORMA DE DESARROLLO SPARK CORE

Spark Core es una plataforma de desarrollo nueva que integra dos subsistemas, un micro-controlador y un módulo de conexión Wi-Fi, diseñado para facilitar la creación de dispositivos electrónicos conectados a internet y que se programa en el lenguaje Wiring, el mismo que utiliza una de las plataformas de desarrollo electrónico más difundidas actualmente, Arduino.

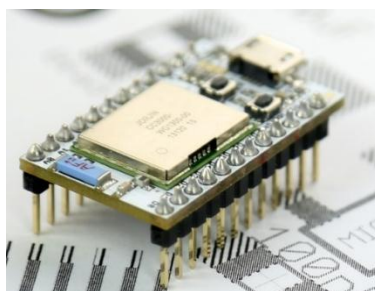


FIGURA 20 SPARK CORE © SPARK TEAM

El hardware que integra la placa del Spark Core es superior a lo que ofrecen diferentes casas fabricantes para el tipo de aplicaciones como la de la presente tesis. Consiste de un micro-controlador STM32F103 el cual es compatible con la arquitectura ARM Cortex M3, entre las características más destacadas se encuentra que es un micro-controlador de 32 bits que trabaja a 72 MHz, internamente posee 128 KB de memoria Flash y 20 KB de RAM.

Adicionalmente al micro-controlador, se encuentra un módulo de conexión Wi-Fi, el cual actualmente es adecuado para dotar de conexión inalámbrica a internet a dispositivos electrónicos que lo requieran de manera muy simple. El módulo es un CC3000 Simple Link de Texas Instruments.

A continuación se describe de manera más extensa tanto el micro-controlador como el módulo Wi-Fi.

3.2.1. Micro – controlador STM32F103

Este micro-controlador producido por el mayor fabricante europeo de semiconductores *STMicroelectronics* o simplemente *ST*, es compatible con la arquitectura ARM ya que su núcleo es justamente un ARM Cortex-M3.

Arquitectura ARM

“ARM (*Advanced RISC Machines*) es una arquitectura *RISC* (*Reduced Instruction Set Computer*) de 32 bits la cual fue desarrollada por la compañía británica ARM Holdings, siendo el conjunto de instrucciones de 32 bits más utilizado” [32]. Entre sus características destaca su bajo consumo por lo que son muy populares para dispositivos móviles (celulares, tablets).

ARM® Cortex™-M3

Es la última generación de procesadores ARM para sistemas embebidos. Ha sido desarrollado para proveer una plataforma de bajo costo que cumple con las necesidades para implementar MCUs, con reducidos número de pines y bajo consumo, ofreciendo además rendimiento computacional excepcional y un avanzado sistema de respuesta a interrupciones.

Características del micro-controlador.

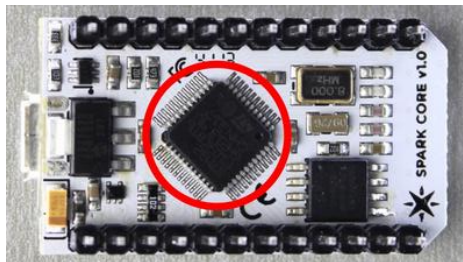


FIGURA 21 STM32F103CBT6 EN SPARK CORE V1.0 © SPARK TEAM

La gama de micro-controladores STM32F103 es amplia, en el caso del micro-controlador del Spark Core, la placa incluye un **STM32F103CBT6** cuyas características son las siguientes y fueron obtenidas del documento de referencia (*datasheet*) del micro-controlador disponible en la página web del fabricante.

- Core (*núcleo*) ARM 32-bits Cortex™-M3 CPU. Máxima frecuencia 72 MHz
- Memoria 64 ó 128 Kbytes de memoria Flash. 20 Kbytes de SRAM
- Alimentación de 2.0 a 3.6 V
- Comunicación: SPI, I2C, USART, USB y CAN
- 3 Timers de propósito general y 1 de control avanzado

3.2.2. Características de Spark Core

Las características mencionadas anteriormente, son referentes al micro-controlador sobre el cual está construido la placa de desarrollo Spark Core,



considerando esto se presentan a continuación las características que están disponibles para ser utilizadas.

- 8 pines Input/Output digitales y 8 pines Input/Output analógicos
- Convertidor Analógico – Digital de 12 bits
- USB 2.0 full-speed
- Comunicación: SPI, UART (Serial), I2C y JTAG
- Programación inalámbrica, vía USB o JTAG
- Alimentación 3.3V DC con regulador on-board que permite alimentar mediante USB Micro
- 100mA de consumo típico, máximo 300mA
- 128KB de memoria flash y 20KB SRAM
- 2MB de memoria flash externa; EEPROM incluida en el módulo Wi-Fi CC3000.

Además de tener 128KB de memoria flash interna para almacenar el firmware, el Spark Core también posee una memoria externa SPI SST25VF016B. La capacidad de la memoria externa es de 2MB y es usada para almacenar el firmware de fábrica que permite reiniciar la placa en caso de cualquier inconveniente.

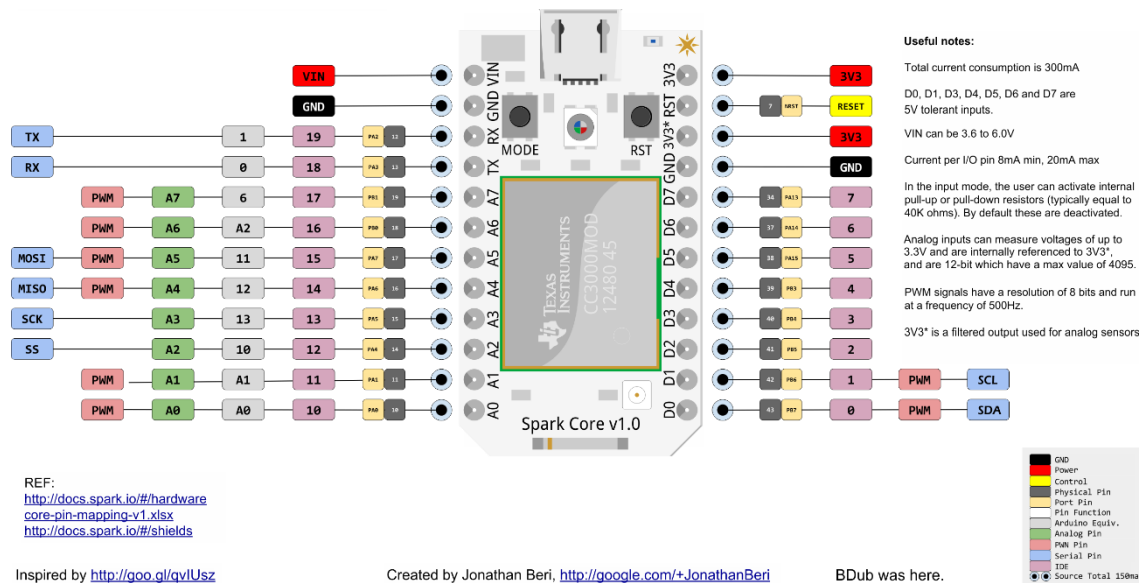
El módulo Wi-Fi on board, se trata del Texas Instruments CC3000, el cual posee las siguientes características:

- IEEE 802.11 b/g
- Rendimiento: Potencia Tx: +18dBm a 11 Mbps y Sensibilidad Rx: -88 dBm
- Temperatura de operación: -20°C a 70°C
- Seguridad: WEP, WPA y WPA2
- Interfaz de comunicación SPI

Disposición de pines

Se tiene un total de 18 pines de entrada o salida disponibles para el usuario, 8 digitales del *D0* al *D7*, 8 analógicos del *A0* al *A7* y dos pines exclusivos para conexión Serial *Tx* y *Rx*. Todos estos pines trabajan a 3.3V con excepción de *D0*, *D1*, *D3*, *D4*, *D5*, *D6* y *D7* que pueden trabajar a 5V por lo que es necesario considerar esto antes de conectar con algún periférico que trabaje con diferente alimentación.

La siguiente figura muestra la disposición de los pines.



3.2.3. Configuración de conexión Wi-Fi

Una de las características más interesantes del Core, es lo simple que resulta conectar el mismo a una red WiFi. Para ello es necesario descargar la aplicación oficial de Spark ya sea para dispositivos con sistema operativo Android o iOS.

Una vez instalada la aplicación, se debe poner al Spark Core en modo de escucha o Listening, en el cual el led RGB de la placa debe destellar de color azul. En caso de no estar en el modo Listening, se debe presionar el botón MODE por 3 segundos servirá para forzar al dispositivo a entrar a dicho modo. También se debe presionar por 10 segundos el botón MODE de la placa hasta que destelle rápidamente de color azul, en caso de que se desee borrar credenciales WiFi anteriores.

Una vez en el modo escucha y con la aplicación de Spark abierta, se debe mostrar la pantalla para ingreso de credenciales de la red WiFi automáticamente, la cual es similar a la que se indica en la siguiente figura.

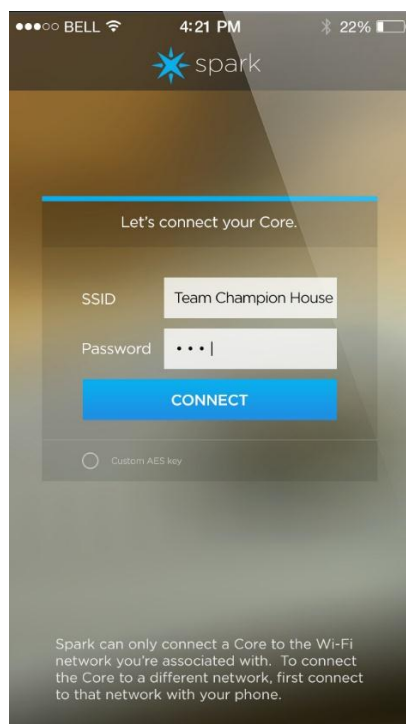


FIGURA 23 APLICACIÓN PARA CONFIGURAR RED WiFi EN SPARK CORE © SPARK TEAM

3.3. LECTOR RFID MFRC522

El lector RFID seleccionado se basa en el uso del integrado MFRC522 de NXP Semiconductors, el cual está diseñado para leer etiquetas RFID creadas bajo el estándar ISO/IEC 14443 A/MIFARE sin necesidad de circuitos integrados adicionales. Como se menciona en el documento de referencia del RC522 [28], provee una implementación eficiente y robusta para demodular y decodificar señales de las tarjetas creadas bajo el estándar antes mencionado y sus versiones compatibles.

El MFRC522 cuenta con las interfaces Serial, SPI y Serial UART para comunicación con otros dispositivos.

El módulo de transmisión del integrado soporta modos de Lectura/Escritura usando varias velocidades de transferencia y protocolos de modulación. Para la etapa de comunicación del lector a la tarjeta se usa modulación ASK, codificación Miller y la velocidad de transferencia es de 106 a 848 KBd¹. La comunicación en sentido contrario es decir, tarjeta a lector es mediante modulación cargada con subportadora², codificación Manchester o BPSK y velocidades de transferencia entre 106 – 848 KBd.

3.3.1. Interfaz de comunicación SPI

¹ La unidad es Kilobaudios

² Este tipo de modulación se la puede encontrar detallada en la referencia [2].



Para la comunicación entre el MFRC522 y un micro-controlador se puede usar SPI, I²C o Serial UART, con la característica de que el integrado reconocerá automáticamente la interfaz utilizada después de realizar un reinicio forzado o hard reset del integrado.

Para la presente tesis se adquirió una placa que incluye el integrado con la interfaz SPI lista para ser utilizada, la cual puede manejar velocidades de hasta 10 Mbits/s. Al comunicarse con el micro-controlador el MFRC522 lo hace en modo esclavo, recibiendo los datos de configuración del integrado y también enviando y recibiendo datos relevantes a la interface de comunicación.

Los bytes de datos en las líneas MOSI y MISO son enviados con el bit más significativo primero lo cual debe ser considerado en la programación al momento de recibir y enviar los mismos desde el micro-controlador. De igual manera los datos en las líneas MOSI y MISO deben ser estables en el flanco de subida del clock el cual es generado por el Master (micro-controlador) y pueden ser cambiados en el flanco de bajada. Los datos son entregados por el MFRC522 en el flanco de bajada y son estables durante el flanco de subida.

Los datos son leídos del MFRC522 mediante la interfaz SPI como se indica en la siguiente tabla, es posible leer n bytes y el primer byte enviado define el modo y la dirección.

MOSI	dirección 0	dirección 1	dirección 2	...	dirección n	dirección n+1
MISO	X ³	dato 0	dato 1	...	dato n-1	dato n

TABLA 2 LECTURA DE DATOS. FUENTE: [28]

El primer byte como ya se mencionó, indica al MFRC522 que el Master desea leer datos, además indica la primera dirección de memoria que desea leer. Como se puede observar los datos están retrasados pues cuando se envía la segunda dirección, el MFRC522 está entregando los datos de la primera lo cual debe ser considerado al momento de programar.

La secuencia de escritura se indica en la siguiente tabla. Es posible escribir n bytes enviando una sola dirección. Nuevamente el primer byte define el modo y la dirección.

³ No importa



MOSI	dirección 0	dato 0	dato 1	...	dato n-1	dato n
MISO	X	X	X	...	X	X

TABLA 3 ESCRITURA DE DATOS. FUENTE [28]

Después de enviar el primer byte que indica el modo y la dirección, se puede enviar en los siguientes los datos que se requiera.

La estructura del byte de dirección (byte 0) es la que se muestra a continuación.

1 = leer 0 = escribir	dirección						0

TABLA 4 BYTE DE DIRECCIÓN. FUENTE [28]

3.3.2. Registros y comandos

Para interactuar con el MFRC522 desde el micro-controlador y poder leer o escribir datos, es necesario utilizar los registros del integrado, así como los diferentes comandos. Dependiendo de la funcionalidad de cada registro, las condiciones de acceso a un registro pueden variar. En principio los bits con el mismo comportamiento están agrupados en registros comunes. Las condiciones de acceso pueden ser:

- R/W: Lectura y escritura. Pueden ser leídos y escritos por el micro-controlador debido a que son utilizados solo para propósitos de control.
- D: Dinámico. Pueden ser leídos y escritos por el micro-controlador sin embargo, también pueden ser escritos automáticamente por ejemplo después de haber ejecutado un comando.
- R: Solo de lectura. Son modificados solo internamente y no pueden ser escritos desde el micro-controlador.
- W: Solo de escritura: Al leer estos registros siempre retorna cero.
- Reservados: Son reservados para uso futuro y no deben ser cambiados.

⁴ Bit más significativo

⁵ Bit menos significativo



- RFT: Reservados para uso futuro o para pruebas de producción, no deben ser cambiados.

Uno de los registros más importantes es aquel que permite la ejecución de comandos, en el datasheet del MFRC522 se lo puede encontrar como CommandReg y tiene la siguiente estructura.

Símbolo:	Reservado	RcvOff	PowerDown	Comando[3:0]
Acceso:	-	R/W	D	D

TABLA 5 REGISTRO COMMANDREG. FUENTE [28]

Los cuatro últimos bits son para el activar un comando de acuerdo al valor respectivo de cada comando. Existe un total de 11 comandos numerados binariamente desde el 0000 (0) hasta el 1111 (15), no se consideran el 0101, 0110, 1001, 1010, 1011. Por ejemplo el comando para transmitir es el 0100 y para activar el circuito de recepción 1000.

3.4. LIBRERÍA PARA MFRC522

En la sección anterior se consideró el funcionamiento de la interfaz SPI del MFRC522, las secuencias de lectura y escritura así como los registros y los comandos respectivos que permiten poner en funcionamiento el integrado. Para comunicarse desde el micro-controlador es necesario programar una librería que permita siguiendo las pautas de la conexión SPI, obtener los datos de cada tarjeta, específicamente su identificador único o UID.

Cada fabricante suele proveer una librería para controlar sus dispositivos, en este caso en Internet se puede encontrar varias librerías desarrolladas por los usuarios pero también una oficial la cual está programada en C51, una variación de C adaptada para los micro-controladores 8051 de Intel, la cual fue tomada como referencia para desarrollar la librería.

Inicialmente se realizaron pruebas con el MFRC522 con una librería que se portó de C51 al lenguaje C del compilador mikroC lo cual permitió comprender de forma más detallada el funcionamiento de los registros y comandos disponibles mediante el uso de un microcontrolador PIC16f887. La librería se encuentra en el anexo del documento.

Una vez realizadas las pruebas en el MFRC522 se portó nuevamente la librería ahora a C++ y se utilizó una placa Arduino la cual es compatible con el Spark Core. Finalmente después de realizar ciertos cambios y considerando la librería disponible para Spark Core se obtuvo una librería adecuada para la aplicación,

sin considerar el acceso a los registros de memoria que no son utilizados en esta aplicación.

A continuación se describen los pasos que deben realizarse para poner en funcionamiento el MFRC522 y el micro-controlador mediante la interfaz SPI la cual debe ser previamente configurada de acuerdo a la distribución de pines de la tarjeta y el micro-controlador. Se recomienda utilizar un módulo SPI simulado por software en pines digitales en lugar de un módulo SPI físico, debido a que en las pruebas realizadas se obtuvieron mejores resultados.

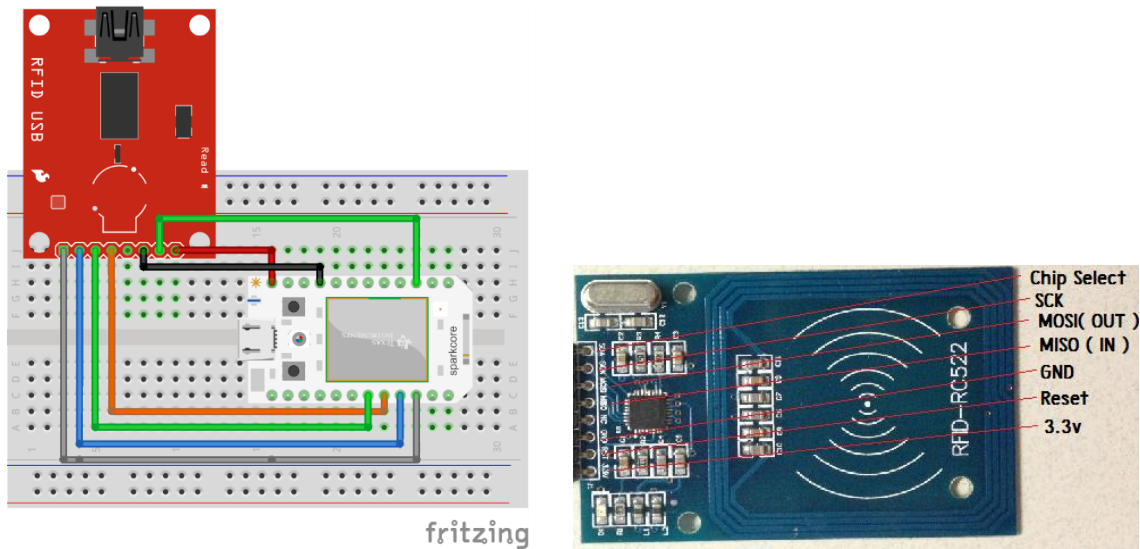


FIGURA 24 CONEXIÓN DE RFID MFRC522 Y SPARK CORE. FUENTE: EL AUTOR

3.4.1. Inicialización

Función: Init()

- Reiniciar el MFRC522 mediante el pin RST
- Configurar el Timer, los valores establecidos en la librería original son respetados. Se deben configurar los registros siguientes:

TModeReg	2A	0x80
TPrescalerReg	2B	0xA9
TReloadRegH	2C	0x03
TReloadRegL	2D	0xE8

TABLA 6 CONFIGURACIÓN DE TIMER. FUENTE: EL AUTOR



- Modulación ASK

TxASKReg	15	0x40

TABLA 7 MODULACIÓN ASK. FUENTE: EL AUTOR

- Encender la antena

TxControlReg	14	0x03

TABLA 8 REGISTRO PARA ENCENDER LA ANTENA. FUENTE: EL AUTOR

3.4.2. Buscar tarjetas

Función: IsNewCardPresent()

- Se envía un comando WUPA, el cual en el estándar ISO 14443 es un comando Wake-Up que “despierta” a las tarjetas cercanas las cuales al recibirlo se preparan para ser seleccionadas. El comando enviado es el 0x52.

De ser exitoso el proceso de buscar tarjetas se continúa al siguiente paso, de lo contrario esta función debe estar en un bucle que se repita hasta recibir una respuesta de confirmación por parte de alguna tarjeta cercana al lector.

3.4.3. Obtener el UID

Función: ReadCardSerial()

- Después de la confirmación del anterior paso, es necesario que la tarjeta que respondió se mantenga en estado activo y las otras vuelvan al estado IDLE o libre.
- Una vez seleccionado el UID devuelto por la tarjeta puede ser de 4, 7 o 10 bytes, dependiendo del tipo de tarjeta MIFARE.
- Si el proceso de recepción del UID es exitoso se pasa al siguiente paso.

3.4.4. Leer el UID



- Es el paso final pues se lee un número fijo de bytes donde cada uno representa una parte del UID de la tarjeta, como ya se mencionó el número de bytes es variable dependiendo del tipo de tarjeta MIFARE. La variable que almacena los bytes es `uidByte[]` y el tamaño del código se lo obtiene de `uid.size`.

3.5. CLIENTE TCP

Una vez ha sido leído el UID de la tarjeta del usuario, es necesario enviarlo al servidor para que se pueda verificar los datos y habilitar el acceso a un aula o laboratorio. Para lograrlo, se utilizó la librería `TCPClient` la cual permite al Spark Core conectarse a una dirección específica de Internet y a un puerto, de esta manera es posible comunicarse con el servidor Apache a través del puerto 80 utilizado por el protocolo HTTP.

Establecida la conexión con el servidor, es posible realizar solicitudes GET o POST, enviando los datos necesarios de acuerdo a la descripción realizada en el capítulo 2.

Haciendo referencia al código de programación, se inicia con la declaración de las variables necesarias:

```
7  TCPClient cliente;  
8  IPAddress server(192, 168, 1, 109);  
9  uint16_t port = 80;
```

FIGURA 25 VARIABLES PARA CREACIÓN DE CLIENTE TCP. FUENTE: EL AUTOR

- `cliente` es de tipo `TCPClient`, la librería que provee un socket TCP que permite conectarse a una dirección y puerto específico.
- `server` es la dirección IP del servidor Apache.
- Debido a que se realizan solicitudes HTTP, el puerto debe ser el 80 asignado a la variable `port`.

Una vez declaradas las variables, se conecta el cliente al servidor.



```
void openSocketTCP(){
    cliente.connect(server, port);
    String m= "Conexión Exitosa a Servidor ";
    String oct1 = String(server[0]);
    String oct2 = String(server[1]);
    String oct3 = String(server[2]);
    String oct4 = String(server[3]);
    String servidor = String(oct1 + "." + oct2 + "." + oct3 + "." + oct4);
    if (cliente.connected()){
        Serial.println(m + servidor);
        Serial.println(WiFi.localIP());
    } else {
        Serial.println("Falló conexión!");
        Serial.println(WiFi.localIP());
    }
}
```

FIGURA 26 FUNCIÓN PARA ABRIR UNA CONEXIÓN AL SERVIDOR. FUENTE: EL AUTOR

- Para conectar con el servidor y el puerto definidos en las variables, se utiliza el método connect con los parámetros server y port.
- Para comprobar que la conexión se realizaba exitosamente, se configuró la librería Serial, la cual permite visualizar los datos que se impriman con el método println.
- Para la visualizar la información se utilizó el software PuTTY, el cual no solo sirve para establecer una conexión Serial sino también es cliente Telnet, SSH y Raw. La configuración para establecer conexión serial con el Spark Core es:
 - **Serial Line:** COM3
 - **Velocidad:** 9600 baudios
 - **Bits de datos:** 8
 - **Bits de parada:** 1
 - **Paridad:** Ninguna
 -

FIGURA 27 VISUALIZACIÓN DE DATOS RECIBIDOS SERIALMENTE. FUENTE: EL AUTOR

Finalmente mediante la conexión ClienteTCP, se realiza la solicitud HTTP que permite enviar el ID del dispositivo al servidor para que sea recibido y procesado por la API.



```
88 void sendToAPIServer(){
89     cliente.flush();
90     delay(1);
91     cliente.println("GET /APITesis/receiveUID.php?myID=" + myID + " HTTP/1.0");
92     Serial.println(myID);
93     cliente.println("Host: 192.168.1.108");
94     cliente.println("Connection: close");
95     cliente.println();
96     delay(10);
97     cliente.flush();
98     delay(10);
99     cliente.stop();
100 }
```

FIGURA 28 CÓDIGO DE SOLICITUD HTTP. FUENTE: EL AUTOR

Como se puede observar en la figura anterior, la solicitud se la construye como se indicó en la sección 2.6.1, línea por línea.

3.6. DISEÑO DEL CIRCUITO IMPRESO O PCB

Como parte de la implementación del prototipo electrónico se incluye el diseño y elaboración de un circuito impreso o PCB. Para el diseño del mismo se ha utilizado el software Eagle, el cual se lo puede considerar un estándar mundial al momento de diseñar PCBs debido a que compañías como Sparkfun o Pololu dedicadas al diseño, fabricación y venta de todo tipo de placas electrónicas, lo utilizan.

La fuente de alimentación de la placa es una batería de 12V, pero directamente con este voltaje solo trabaja el circuito que controla la cerradura. Para alimentar al Spark Core se utiliza un regulador LM7805 el cual entrega 5V directamente a un conector USB Tipo A, por otra parte el lector RFID trabaja con 3.3V los cuales son entregados directamente por el Spark Core en los pines correspondientes.

Para controlar la cerradura es necesario solo activar la misma con un pulso de breve duración que permite accionar la parte mecánica interna. Se utiliza un circuito clásico que permite separar el circuito alimentado por los 12V de la parte alimentada con 5V, utilizando un relé controlado desde el Spark Core mediante un transistor como se puede observar en la figura correspondiente.

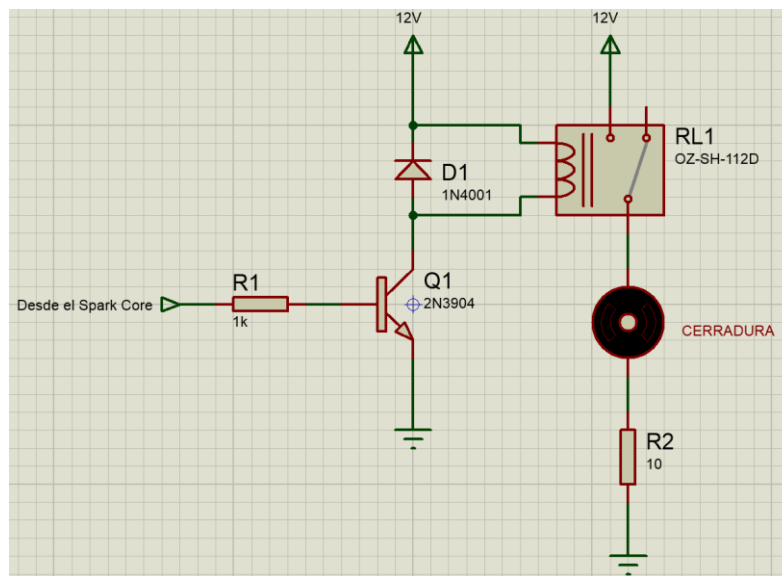


FIGURA 29 CIRCUITO DE CONTROL DE CERRADURA. FUENTE: EL AUTOR

Las siguientes figuras corresponden al esquemático completo del circuito y al PCB con los elementos ya posicionados.

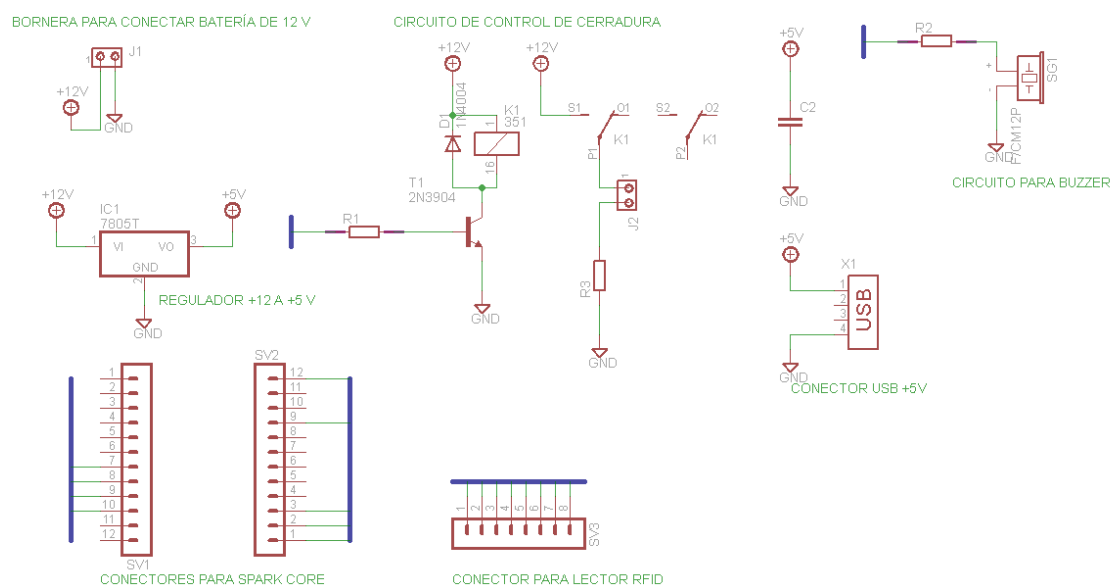


FIGURA 30 ESQUEMÁTICO DEL PCB. FUENTE: EL AUTOR

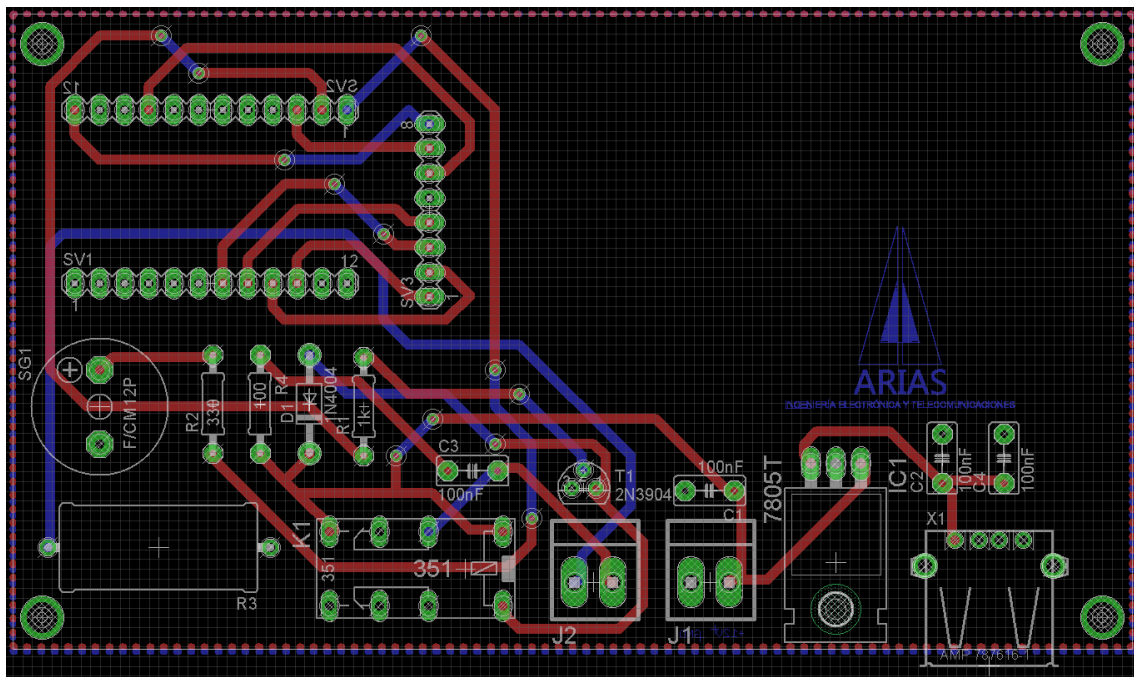


FIGURA 31 DISEÑO DE CIRCUITO IMPRESO O PCB. FUENTE: EL AUTOR

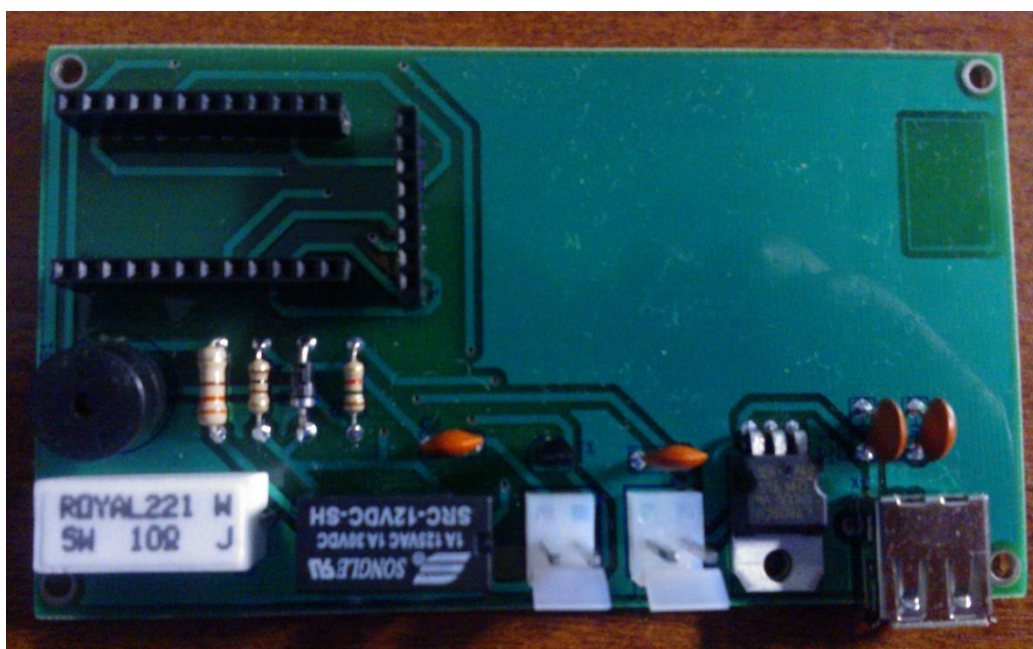


FIGURA 32 PCB IMPLEMENTADO A DOBLE CAPA. FUENTE: EL AUTOR

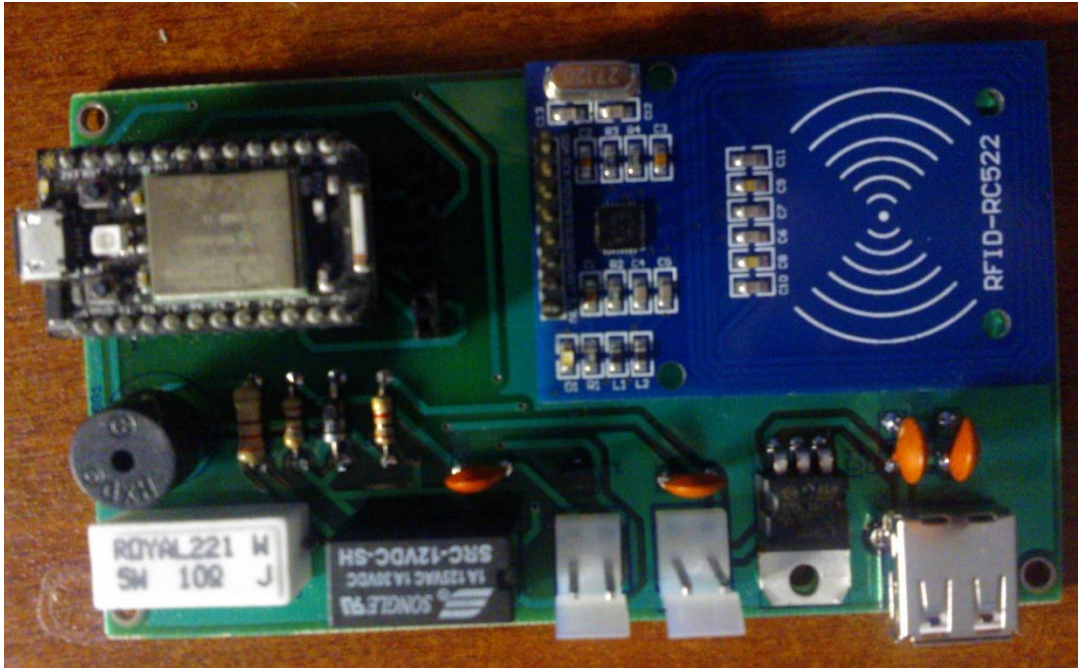


FIGURA 33 PCB IMPLEMENTADO A DOBLE CAPA CON ELEMENTOS POSICIONADOS Y SOLDADOS. FUENTE: EL AUTOR

3.7. COSTOS

Se detalla a continuación el costo de implementación del PCB para el prototipo así como los elementos que lo componen.

Ítem	Cantidad	Precio
Lector RFID RC522 + Tags	1	15,00
Spark Core	1	48,00
Conector USB Type A	1	1,00
Buzzer	1	1,00
Resistencia 10Ohm 5W	1	1,25
Relé 12VDC 1A	1	1,30
LM7805	1	1,50
Resistencias varias	3	0,15
Diodo 1N4148	1	0,25
Capacitores cerámicos 104	3	0,30
Headers hembra para PCB	2	2,50
Conectores macho para PCB	2	2,00
Placa doble capa, antisolde	1	30,00
TOTAL		\$104,25

TABLA 9 COSTOS TARJETA ELECTRÓNICA



CAPÍTULO IV

4. APLICACIÓN WEB DEL SISTEMA

La aplicación web del sistema permite administrar aulas de la facultad a las cuales se las relaciona con el identificador del prototipo instalado, además permite gestionar los usuarios y sus códigos asignados de acuerdo a la tarjeta magnética correspondiente. Se ha considerado que una aplicación web es más adecuado que un programa cliente que debe ser instalado en cada computadora, ya que se utiliza al navegador web como cliente ligero para acceder a la aplicación que reside en un servidor web. Con esto además se evita que el sistema operativo sea un obstáculo para utilizar la herramienta ya que solo se necesita una conexión a internet y un navegador para acceder a la aplicación.

La aplicación se ejecuta sobre un servidor **HTTP Apache** y utiliza como Sistema Gestor de Base de Datos (SGBD) a **mySQL**, el lenguaje en el que está escrito la aplicación es **PHP** sobre el framework de desarrollo **CakePHP**. Cada uno de estos componentes se describe en el presente capítulo así como la lógica de funcionamiento de la aplicación.

4.1. SERVIDOR HTTP

La palabra servidor hace referencia tanto a un software como al hardware donde éste se ejecuta. Un programa servidor es capaz de aceptar solicitudes enviadas por clientes quienes solicitan un recurso a través de un puerto específico y dar respuesta a las mismas basando su funcionamiento en el modelo o arquitectura Cliente – Servidor.

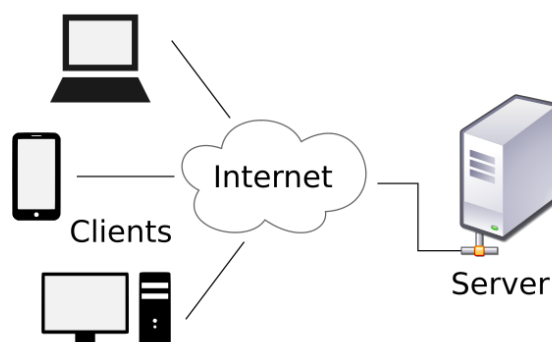




Fig. 12. Arquitectura Cliente – Servidor.

Los servidores pueden proveer diferentes servicios de acuerdo a la aplicación y protocolo para el cual fueron creados, en el caso de un servidor *HTTP*, recibe y responde solo solicitudes basadas en dicho protocolo y que por lo general proviene de clientes conocidos como *navegadores* o *web browsers*, aunque no es estrictamente necesario.

Existen varios servidores *HTTP* disponibles para los diferentes sistemas operativos, algunos de los más conocidos son: Apache HTTP Server, Microsoft IIS, Nginx y Google GWS.

En la presente tesis se ha seleccionado **Apache HTTP Server** como servidor debido a que es software libre y es el más difundido de los mencionados anteriormente. Dado que se trata de un servidor HTTP, Apache necesita que tanto el puerto 80 (HTTP) como el puerto 443 (HTTPS) estén habilitados para el correcto funcionamiento del sistema completo.

4.2. MySQL

MySQL es el Sistema Gestor de Base de Datos relacional, multi – hilo y multi – usuario más popular, es software libre bajo un esquema de licenciamiento dual que obliga a comprar una licencia específica en caso de ser usado en productos comerciales privados. En la actualidad es desarrollado, distribuido y brinda soporte a través de la corporación Oracle.

Como Servidor de Base de Datos, MySQL se caracteriza por ser muy seguro, escalable y fácil de utilizar. Un servidor de BD MySQL, puede correr fácilmente en una PC o computadora portátil, en conjunto con otras aplicaciones y servidores web, como en el caso de la presente tesis.

El software MySQL es un sistema basado en la arquitectura cliente – servidor pues consiste de un servidor SQL *multi – thread* o multi – hilo, es decir soporta múltiples conexiones a la vez a través de varios programas clientes y librerías, herramientas administrativas y un amplio número de APIs (*Application Programming Interfaces*) o interfaces de programación de aplicaciones lo que permiten a aplicaciones escritas en diferentes lenguajes, tener acceso a las bases de datos MySQL. Entre los lenguajes que pueden acceder a MySQL mediante APIs se encuentran C y sus variaciones C++ y C#, Perl, PHP, Python, Java, por nombrar solo algunos de los más difundidos. Además es posible comunicarse con MySQL mediante una interfaz ODBC desde los lenguajes que soporten dicha interfaz.



4.3. PHP

“PHP (*Hypertext Preprocessor*) es un *lenguaje de scripting*⁶ de propósito general que está adecuado especialmente para desarrollo web y además puede ser embebido dentro de *HTML*⁷” [19].

Al decir que puede ser embebido en HTML, se refiere a la posibilidad de usar el lenguaje PHP dentro del código HTML, lo cual permite realizar diferentes tareas mediante el uso de este lenguaje. La palabra *preprocessor* se debe a que PHP hace cambios antes de que la página HTML sea creada. El código PHP está delimitado por etiquetas o instrucciones especiales de inicio `<?php` y fin `?>` que permiten “saltar” dentro y fuera del “modo PHP”. Un ejemplo de código PHP embebido en HTML se muestra en la siguiente figura:

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="UTF-8" />
    <title> Ejemplo básico PHP</title>
  </head>
  <body>
    <?php
      echo 'Hola mundo';
    ?>
  </body>
</html>
```

FIGURA 34 CÓDIGO PHP EMBEBIDO EN CÓDIGO HTML. FUENTE [22]

PHP se caracteriza por ser *server – side* es decir, el código es ejecutado en el servidor el cual posee un intérprete de código PHP, generando HTML el cual es enviado al cliente. El cliente recibe los resultados de haber ejecutado el script, pero no conoce el código subyacente.

Las razones para justificar la elección de PHP como lenguaje para desarrollar la aplicación web y además la *API* del sistema la cual será detallada más adelante, se han considerado son las siguientes:

- PHP puede correr en varias plataformas: Windows, Linux, Unix, OS X.
- Es compatible con casi todos los servidores usados actualmente, entre los cuales se incluye Apache HTTP Server.

⁶ Lenguaje de scripting: es un lenguaje de programación que soporta scripts, programas escritos para un ambiente *run-time* especial que los puede interpretar y automatizar la ejecución de tareas que podrían ser ejecutados uno por uno, por un operador humano [21].

⁷ HTML: Lenguaje de marcado para la elaboración de páginas web [20].



- PHP provee conexión a bases de datos, por supuesto MySQL es una de éstas.
- PHP es software libre, simple y rápido de aprender, además de ser eficiente considerando que se ejecuta del lado del servidor.

Entre las capacidades que PHP ofrece, de acuerdo a la documentación encontrada en el sitio web oficial (www.php.net) y en otros como (www.w3schools.com), se mencionan las siguientes:

- Generación de contenido dinámico para páginas web.
- PHP permite crear, abrir, leer, escribir, borrar, y cerrar archivos en el servidor.
- Se puede recibir datos a través de formularios.
- Envío y recepción de cookies.
- En cuanto a base de datos se refiere, es posible agregar, borrar y modificar datos.
- Permite implementar control de acceso a los usuarios.
- PHP puede encriptar datos.

Al momento de escribir el presente documento, la versión actual de PHP es la 5.6.5.

4.4. XAMPP

Hasta el momento, se han descrito las herramientas que permiten desarrollar la aplicación web para la presente tesis: Apache Web (HTTP) Server, MySQL y PHP.

Sin embargo, para desarrollar la aplicación web se ha optado por no instalar las herramientas por separado sino más bien instalar un entorno de desarrollo que incluye las mismas y que simplifica el proceso de instalación, evitando el hecho de configurar la interacción entre las mismas.

El entorno de desarrollo es **XAMPP**, sus siglas se derivan por las herramientas que incluye el entorno: **X** leído como “cross” debido a que es cross – platform o multiplataforma, **A** por Apache, **M** por incluir a MySQL como SGBD, **P** por el intérprete de PHP y **P** por el intérprete del lenguaje PERL, éste último no se consideró para desarrollar la aplicación y se omitió su instalación.

La instalación del entorno es simple y guiada por el programa de instalación, no se requiere configuración por consola, todo el proceso es realizado de forma gráfica. Una vez que la instalación concluye se puede acceder al panel de configuración mediante la dirección:

`http://localhost/xampp`

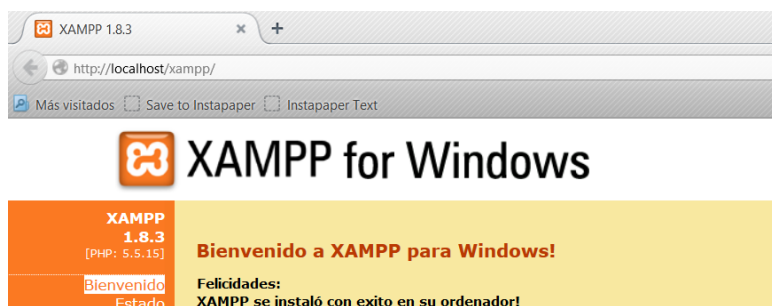


FIGURA 35 PÁGINA DE XAMPP INSTALADO EN WINDOWS. FUENTE: EL AUTOR

Una vez ha sido instalado el entorno de desarrollo XAMPP, cada vez que se requiera iniciar el servidor web y el servidor de base de datos, se lo puede realizar mediante el panel de control de XAMPP, el cual se lo puede hallar en el menú inicio de Windows o en la carpeta de instalación: C:\xampp en donde se encuentra el programa xampp control.

La siguiente figura muestra una captura de pantalla del panel de control:



FIGURA 36 PANEL DE CONTROL DE XAMPP. FUENTE: EL AUTOR

4.5. DISEÑO Y CREACIÓN DE LA BASE DE DATOS

Para el presente proyecto de tesis, contar con una base de datos es importante debido a que permite almacenar información necesaria de los docentes a los cuales se les asigna un nombre de usuario, contraseña y UID (Identificador único de la tarjeta magnética). De igual manera dentro de la base de datos se almacenaran los dispositivos disponibles y las aulas en las que se encuentran instalados. Adicionalmente, se cuenta con un registro de ingreso en el cual cada tupla lo conforman la fecha de ingreso, el aula y el identificador de la persona. La información almacenada será consultada constantemente para el proceso de verificación de datos al momento de solicitar acceso a una de las aulas o laboratorios que cuenten con el prototipo instalado.

El diseño de la base de datos y las relaciones existentes entre las tablas es simple debido a que el sistema tiene un solo objetivo lo cual deriva en la



existencia de unas cuantas entidades. A continuación se presenta un diagrama entidad – relación que permite interpretar de mejor manera la estructura que tiene la base de datos.

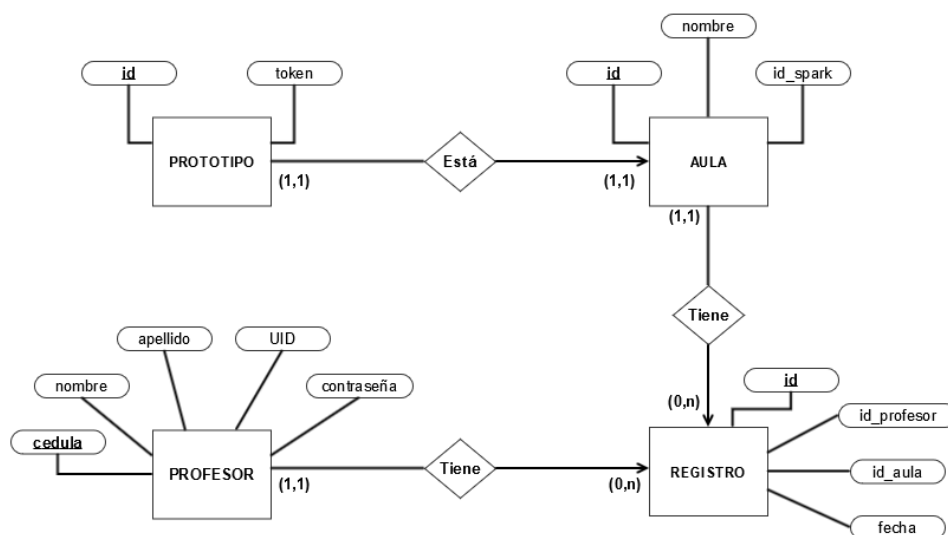


FIGURA 37 DIAGRAMA ENTIDAD – RELACIÓN. FUENTE: EL AUTOR

Del diagrama Entidad – Relación, es posible crear el esquema de la base de datos y las tablas respectivas. Una forma adecuada de representar las tablas es como sigue, mediante un diagrama UML.

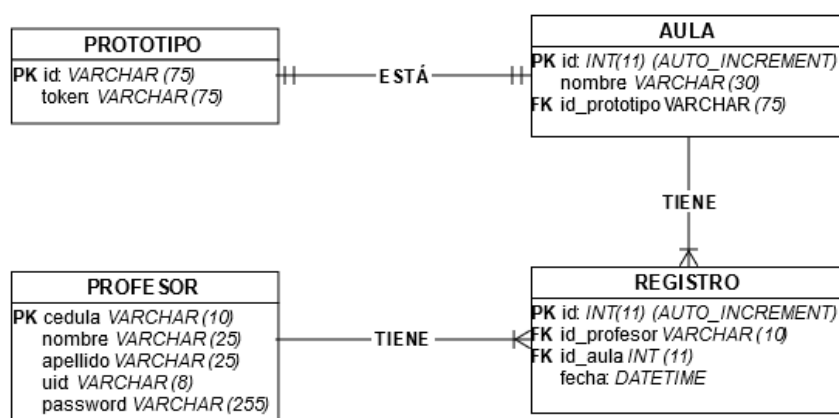


FIGURA 38 DIAGRAMA UML DE LA BASE DE DATOS. FUENTE: EL AUTOR

La creación de la base de datos y las tablas correspondientes se lo puede hacer de forma gráfica mediante el uso de algún programa cliente como *phpmyadmin*, el cual se instala con el entorno XAMPP, aunque existen varias opciones diferentes.



Dentro del entorno de MySQL ya sea de forma gráfica o mediante la consola, se procede a crear la nueva base de datos y las respectivas tablas. A continuación las figuras muestran las diferentes tablas creadas para la base de datos llamada tesis, de acuerdo a los diagramas antes presentados:

- Profesores

Field	Type	Null	Key	Default	Extra
cedula	varchar(10)	NO	PRI	NULL	
name	varchar(25)	NO		NULL	
last_name	varchar(25)	NO		NULL	
uid	varchar(8)	NO	UNI	NULL	
password	varchar(255)	NO		NULL	

FIGURA 39 TABLA PROFESORES. FUENTE: EL AUTOR

- Dispositivos (Prototipo)

Field	Type	Null	Key	Default	Extra
id	varchar(75)	NO	PRI	NULL	
token	varchar(75)	NO		NULL	

FIGURA 40 TABLA DISPOSITIVOS. FUENTE: EL AUTOR

- Aulas

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
name	varchar(30)	NO		NULL	
id_device	varchar(75)	NO	UNI	NULL	

FIGURA 41 TABLA AULAS. FUENTE: EL AUTOR

- Registros

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
id_teacher	varchar(10)	NO	MUL	NULL	
id_room	int(11)	NO	MUL	NULL	
fecha	datetime	NO		CURRENT_TIMESTAMP	

FIGURA 42 TABLA REGISTROS. FUENTE: EL AUTOR

Una vez creada la base de datos, es posible desarrollar una aplicación web CRUD, éste término hace referencia en programación a un software que



permite las siguientes operaciones: **Create** (crear), **Read** (leer, obtener), **Update** (actualizar) y **Delete** (eliminar).

La aplicación web ha sido desarrollada en lenguaje PHP, pero sobre el framework de desarrollo *CakePHP*.

4.6. CakePHP

Un framework de desarrollo se lo define como “una abstracción de la cual un software provee funcionalidades genéricas las cuales pueden ser modificadas mediante la adición de código escrito por el usuario del mismo, de esta manera se provee un software específico para cierta aplicación” [23].

Específicamente CakePHP en su página web oficial es definido como “una estructura fundamental que sirve de base a los programadores para que éstos puedan crear aplicaciones web. El objetivo principal es permitir al desarrollador, trabajar de forma estructurada y rápida sin perder flexibilidad” [24].

Básicamente el framework de desarrollo evita tener que empezar con un proyecto en PHP desde cero, además de que garantiza que el núcleo de la aplicación (el framework en sí mismo) se mejora constantemente y además ha sido probado de manera rigurosa.

Un ejemplo de las ventajas de utilizar un framework es la capacidad de evitar la creación de conexiones a la base de datos desde código, simplemente es necesario configurar los parámetros de acceso a la misma y el framework proporciona el acceso a la misma de forma segura, además ya no se requiere el uso de comandos SQL directamente en el código para crear, obtener, actualizar o eliminar información de la base de datos debido a que, el framework provee de métodos o funciones que permiten dichas operaciones. La ventaja de hacerlo de esta manera es que se trabaja de forma estructurada y segura sin necesidad de estar creando un proyecto desde cero sino más bien dedicándose a generar la lógica que permita con las herramientas que entrega el framework, alcanzar el objetivo de la aplicación web.

Entre las características que CakePHP ofrece al usuario y que se han considerado importantes para la presente tesis, se encuentran las siguientes:

- CRUD integrado para la interacción con bases de datos, entre las cuales se encuentra MySQL.
- Generación de código.
- Arquitectura Modelo Vista Controlador.
- Validación integrada.
- Plantillas rápidas y flexibles.



- Ayudantes o *helpers* para AJAX, Javascript, formularios HTML, etc.
- Listas de control de acceso.
- Funciona en cualquier subdirectorio del sitio web con poca o ninguna configuración de Apache Web Server.

4.6.1. Arquitectura: Modelo Vista Controlador (MVC)

Es una arquitectura diseñada en los años 60 que en la actualidad es muy utilizada en el desarrollo web. La idea de la arquitectura o patrón de desarrollo MVC es separar los datos, la lógica de negocios y las interfaces de usuarios en una aplicación; a partir de ésta separación, surgen los tres componentes o capas: modelo, vista y controlador.

De forma muy breve se explica a continuación cada parte de la arquitectura, aunque es ciertamente simple cuando se comprende, hacerlo es algo complicado en un principio pues es un paradigma diferente de programación del que se acostumbra.

Modelo

El modelo es la capa encargada de los datos e implementa la lógica de negocio, básicamente posee mecanismos para acceder y actualizar la información. En el caso del presente proyecto, se encarga de comunicarse con la base de datos mediante funciones que acceden a la información contenida en tablas y realizan las funciones CRUD habituales, lo cual incluye además el procesamiento, validación y asociación de los datos.

Vista

Es la capa encargada de la presentación de los datos proporcionados por la capa modelo, en el caso de la presente tesis, la capa vista presenta los datos en lenguaje HTML, aunque no se limita a éste último sino más bien puede ofrecer varios formatos en función de las necesidades de la aplicación como música, documentos, videos, etc.

Controlador

La capa controlador funciona como enlace entre las capas vista y modelo debido a que es la encargada gestionar las peticiones de los usuarios y enviar los comandos a la capa modelo para actualizar los datos, y a su vez a la capa vista para modificar la presentación de los mismos al usuario. La capa controlador no puede manipular los datos ni generar una salida de los mismos para ser mostrados al usuario.



4.6.2. Ruteo

El ruteo es una característica de CakePHP que mapea URLs a las acciones o funciones de los controladores. Se puede acceder a una acción directamente mediante la URL, ingresando su nombre en la solicitud, de igual manera se puede pasar parámetros a las acciones del controlador en la misma URL. La estructura que se obtiene es como la que sigue:

`http://ejemplo.com/controlador/acción/param1/param2/param3`

Se puede definir rutas personalizadas lo que genera URLs limpias y simples para el usuario. CakePHP ofrece documentación completa y simple de comprender acerca de esta y otras características del framework.

4.7. DISEÑO DE LA APLICACIÓN WEB

La aplicación web es la herramienta que permite al administrador, registrar a las personas que tendrán acceso a las aulas o laboratorios en los que se instale el prototipo. Por lo tanto también permite administrar los dispositivos prototipos disponibles, así como las aulas existentes.

4.7.1. Ingreso (Login)

Para acceder a la aplicación se ha creado una página inicial en la que el usuario deberá ingresar credenciales válidas, las cuales debieron ser previamente entregadas. El usuario es en éste y todos los casos, el número de cédula de cada individuo debido a que es un identificador único. Se ha implementado autenticación⁸ de manera que cualquier usuario que requiera ingresar a la aplicación, necesariamente debe proporcionar las credenciales de administrador, de esta manera es posible identificar, autenticar y autorizar usuarios lo cual es común en casi cualquier aplicación web.

⁸ Autenticación: es el proceso de identificar usuarios mediante credenciales (usuario y contraseña) de manera que se asegura que el usuario es quien dice ser [27].



CakePHP: the rapid development php framework

SISTEMA DE GESTION DE ACCESO A LAS AULAS DE LA FACULTAD DE INGENIERIA

Usuario

Contraseña

INGRESAR

FIGURA 43 PÁGINA DE LOGIN A LA APLICACIÓN. FUENTE: EL AUTOR

Como se puede observar en la figura que indica la página de login, la URL en la barra de direcciones es del tipo que se indicó anteriormente en el tema de ruteo.

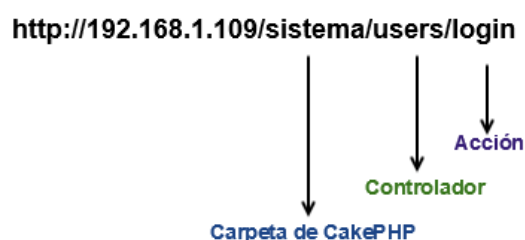


FIGURA 44 URL DE LOGIN

La aplicación está basada en la arquitectura MVC, por lo tanto existe código para cada una de las partes. El código completo de la aplicación se encuentra en el apéndice respectivo del presente documento, pero en este caso se explicará brevemente el código del controlador y la vista de la acción login con el objetivo de sustentar al lector el funcionamiento.



Controlador

```
1 <?php
2 class UsersController extends ApplicationController{
3     public $helpers = array('html', 'Form');
4     public $components = array('Session');
5
6     public function index(){
7
8     }
9     public function login(){
10         if ($this->request->is('post')){
11             if ($this->Auth->login()){
12                 return $this->redirect($this->Auth->redirect());
13             }
14             else{
15                 $this->Session->setFlash('Usuario o contraseña no validos');
16             }
17         }
18     }
19
20     public function logout(){
21         if($this->Auth->logout()){
22             return $this->redirect($this->Auth->redirect());
23         }
24     }
25 }
26 }
```

FIGURA 45 CONTROLADOR DE USUARIOS. FUENTE: EL AUTOR

La clase `UsersController` hereda de la clase `AppController` que es el controlador base del framework. Se hace uso de helpers los cuales facilitan el uso en este caso de formularios HTML en los cuales el usuario ingresa las credenciales.

Se tiene dos funciones importantes, la de ingreso o login y la de salida o logout las cuales se sobrentiende su funcionamiento. Como se puede observar en el código de la función login se comprueba que las credenciales sean enviadas mediante un método POST el cual es generado por el botón INGRESAR de la página. Así mismo se puede observar que se comprueba que la autenticación sea exitosa, esto en la línea de código 11. Finalmente en caso de que las credenciales sean válidas, se re-direcciona a otra página la cual solo es accesible después de autenticarse con el sistema.

De esta manera es posible demostrar las ventajas de utilizar el framework en lugar de empezar un proyecto desde cero, pues permite que el desarrollador se enfoque en la lógica de funcionamiento ya que en ningún momento se ha realizado consultas directamente a la base de datos para comprobar las credenciales, esto se lo ha realizado con las funciones que posee el framework las cuales deben ser configuradas por el desarrollador para adaptarlas a las necesidades de la aplicación. Al usar este modelo de programación estructurada, se garantiza el hecho de que las funciones que se utilizan han sido probadas lo



suficiente como para garantizar un acceso seguro y eficiente a la información en la base de datos.

Vista

```

1 <h2>SISTEMA DE GESTION DE ACCESO A LAS AULAS DE LA FACULTAD DE INGENIERIA</h2>
2
3 <?php
4     echo $this->Session->flash('auth');
5     echo $this->Form->create('User');
6     echo $this->Form->input('username', array('label'=>'Usuario'));
7     echo $this->Form->input('password', array('label'=>'Contraseña'));
8     echo $this->Form->end('INGRESAR');
9 ?>

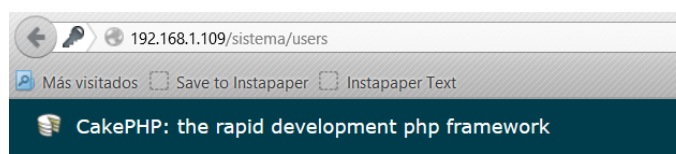
```

FIGURA 46 VISTA DE LOGIN. FUENTE: EL AUTOR

Como se puede observar, el código de la vista de la página login, es decir lo que se presenta al usuario mediante el navegador, es muy simple. Básicamente se trata de un archivo que posee en su mayoría código PHP y algo de HTML. La línea 1 muestra un título en la página web y está escrito en HTML, mientras que desde la línea 3 a la 9 se identifica código PHP el cual simplemente está mostrando al usuario los controles del formulario como cajas de texto y un botón llamado ingresar. La línea 4 y 5 son de configuración de sesión para verificar las credenciales del administrador en el controlador.

4.7.2. Dashboard de la aplicación

Es la página inicial que el administrador visualiza después de haberse autenticado con el sistema. Muestra las acciones administrativas que se puede realizar, básicamente dar de alta o de baja a profesores, prototipos y aulas.



[Administrar Profesores](#)

[Administrar Prototipos](#)

[Administrar Aulas](#)

FIGURA 47 DASHBOARD DEL ADMINISTRADOR. FUENTE: EL AUTOR

4.7.3. Administrar profesores, prototipos y aulas

Son las páginas implementadas para realizar las operaciones CRUD a la información almacenada en la base de datos.



Profesores

Agregar Profesor

Cedula	Nombre	Apellido	Acciones
3333333333	SOFIA	OCHOA	Editar Eliminar
1111111111	KARINA	VILLA	Editar Eliminar
0102030405	JUAN	PEREZ	Editar Eliminar
0106551419	GABRIELA	BULGARIN	Editar Eliminar
2222222222	EDGAR	JERVES	Editar Eliminar
9999999999	DIEGO	JARRIN	Editar Eliminar
0104741723	CHRISTIAN	ARIAS	Editar Eliminar
4444444444	ANDRES	CARPIO	Editar Eliminar

FIGURA 48 PÁGINA PARA ADMINISTRAR PROFESORES. FUENTE: EL AUTOR

Las funciones que se han implementado para profesores, prototipos y aulas son: agregar o crear, editar, eliminar y asignar en el caso de las aulas debido a que al crear una aula se le debe asignar un dispositivo instalado en la misma.

Agregar Profesor

Cedula*

Nombre*

Apellido*

UID*

Password*

FIGURA 49 FORMULARIO PARA AGREGAR UN NUEVO PROFESOR. FUENTE: EL AUTOR

4.8. INSTALACIÓN DE LA APLICACIÓN WEB

De esta manera se ha implementado la aplicación web, la cual posee todas las funciones necesarias para cumplir los objetivos en cuanto a la creación de un software que permita administrar el sistema. La elección del lenguaje PHP y el framework CakePHP se debe a que se puede aprender en poco tiempo el lenguaje, facilitando la implementación de una aplicación web sin necesidad de requerir amplios conocimientos de programación.

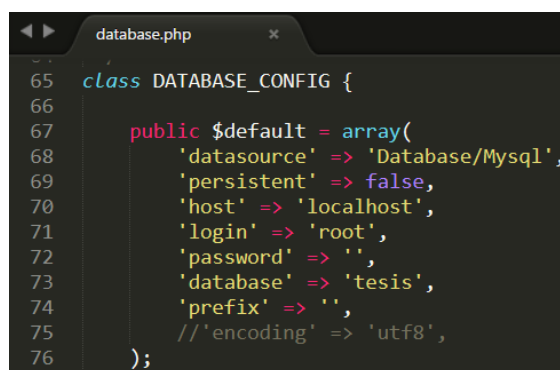


En cuanto a la instalación de la misma, se requiere tener instalado XAMPP, que como se mencionó anteriormente incluye el servidor HTTP Apache, MySQL como SGBD y el intérprete de PHP. El framework CakePHP no es más que un archivo que se lo descarga de la página oficial del mismo (www.cakephp.org) y se lo copia a la carpeta htdocs del directorio C:/xampp.

Una vez copiados los archivos es necesario configurar ciertos archivos que permitirán conectar que el framework trabaje con las herramientas instaladas con XAMPP.

En el directorio de CakePHP el cual ha sido renombrado como sistema, existe un archivo llamado database.php el cual debe ser modificado. Para hacerlo se abre el archivo con cualquier editor de texto como el bloc de notas de Windows, en este caso se utilizó Sublime Text 3 que es un editor de texto muy potente que reconoce la sintaxis del documento dependiendo del lenguaje en el que está escrito. El archivo database.php se encuentra en la dirección: C:\xampp\htdocs\sistema\App\Config o similar dependiendo si se cambiaron las rutas de instalación de XAMPP.

Una vez abierto el archivo, el cual tiene varios parámetros de configuración, se debe ubicar la clase DATABASE_CONFIG y modificar la variable \$default que contiene los parámetros de conexión a la base de datos. En la figura correspondiente se puede observar la configuración para trabajar con la base de datos del presente proyecto la cual fue llamada tesis.



```
65 class DATABASE_CONFIG {
66
67     public $default = array(
68         'datasource' => 'Database/Mysql',
69         'persistent' => false,
70         'host' => 'localhost',
71         'login' => 'root',
72         'password' => '',
73         'database' => 'tesis',
74         'prefix' => '',
75         //'encoding' => 'utf8',
76     );
77 }
```

FIGURA 50 CONFIGURACIÓN DE LA CONEXIÓN DE CAKEPHP CON MYSQL. FUENTE: EL AUTOR

Como se puede observar en la figura, entre los parámetros se especifica conexión a MySQL, el host en este caso es localhost pues tanto MySQL como CakePHP están instalados en el mismo computador. Adicionalmente se configura el usuario y la contraseña, así como el nombre del esquema de base de datos que se utiliza.

Además del archivo de configuración de la base de datos, se debe modificar el archivo core.php en el cual están cadenas de caracteres conocidos como Salt



y Cipher Seed los cuales son utilizados cuando se desea encriptar contraseñas mediante algún método criptográfico como *SHA1* o *MD5*, entre otros. En caso de utilizar alguna encriptación que use estos valores, es necesario guardarlos de forma segura como respaldo. En el caso de la presente tesis, las contraseñas si son encriptadas pero con un método criptográfico más seguro el cual genera un Salt aleatorio cada vez para cada contraseña. El archivo `core.php` se encuentra en la ruta: `C:\xampp\htdocs\cakephp\App\Config` y es necesario modificarlo aun así no se use los valores debido a que hasta que no se lo haga, el framework mostrará notificaciones indicando que se lo debe hacer por motivos de seguridad.

```
223  /
224  /* A random string used in security hashing methods.
225  */
226  Configure::write('Security.salt', 'DYhG11b...');
227  /**
228  * A random numeric string (digits only) used to encrypt/decrypt strings.
229  */
230  Configure::write('Security.cipherSeed', '768...');
231
```

FIGURA 51 CONFIGURACIÓN DEL ARCHIVO CORE.PHP. FUENTE: EL AUTOR

Una vez se termine de realizar las configuraciones, la aplicación esta lista para ser utilizada. Se puede acceder a ella desde cualquier navegador mediante la siguiente dirección la cual incluye la dirección IP del host en el que se tiene instalado XAMPP y el framework.

`http://[ip_host]/sistema/users/login`

En caso de omitir en la dirección la acción `login` o se intente acceder a la fuerza probando otras direcciones, el usuario será re-direccionado a la página de login debido a que si no se autentica con sus credenciales, no podrá acceder.



CAPÍTULO V

5. APLICACIÓN MÓVIL

Como parte del presente proyecto se ha considerado la creación de una aplicación móvil (App) que servirá al igual que las tarjetas magnéticas, como una llave electrónica. Debido a que con la presente tesis además de brindar seguridad a las aulas o laboratorios que lo requieran se pretende también tener fluidez en el desarrollo de las horas de clase, disminuyendo el tiempo de espera en la apertura de aulas por parte del personal de conserjería y en el mejor de los casos evitando su asistencia para liberar al personal a otras tareas, el uso de las tarjetas magnéticas como llaves electrónicas ayuda a solventar el problema pero hay que considerar que actualmente la tendencia electrónica sobre todo la enfocada a lo que se conoce como internet de las cosas, va de la mano con el uso de aplicaciones ya sea para teléfonos inteligentes o tablets con conexión a internet. Es por esta razón que se ha implementado una aplicación móvil en Android, el sistema operativo para smartphones y tablets más difundido en la actualidad incluso sobre el sistema operativo de Apple, iOS. La aplicación tiene conexión con el servidor HTTP Apache al igual que la aplicación web de administración, para la interacción con la base de datos se hace uso de una API desarrollada específicamente para este proyecto en lenguaje PHP.

5.1. INTRODUCCIÓN A ANDROID

Para desarrollar aplicaciones, Android provee de un framework de desarrollo que permite crear aplicaciones para dispositivos como smartphones o tables usando como lenguaje de programación a Java. En internet existen gran cantidad de tutoriales y cursos gratuitos o de pago que permiten aprender a programar aplicaciones para Android, además de todo el material oficial que se puede encontrar en la sección desarrollo de la página oficial del sistema operativo (developer.android.com).

Como todo framework de desarrollo, posee ciertas características que deben ser consideradas para comprender la lógica de funcionamiento del mismo y poder desarrollar más fácilmente aplicaciones, sin mencionar que lo ideal es tener conocimientos básicos acerca del lenguaje Java lo que ayuda a que la curva de aprendizaje sea menor.



5.1.1. Fundamentos de una aplicación.

Las aplicaciones son desarrolladas en Java utilizando ya sea Eclipse o Android Studio, siendo éste último el IDE (Integrated Development Enviroment o Entorno de Desarrollo Integrado) oficial.

Android es un sistema operativo basado en Linux con la característica de ser multi – usuario en la cual cada aplicación es un diferente usuario.

Cada proceso posee una máquina virtual independiente lo cual permite que el código de una aplicación se ejecute aislado de las otras aplicaciones. Predeterminadamente cada App corre en su propio proceso Linux y además tiene acceso solo a los componentes que requiere para realizar su trabajo. En caso de que se necesite compartir datos entre aplicaciones, es posible compartir el mismo ID Linux para que cada una pueda acceder a los archivos de las otras.

5.1.2. Componentes de una aplicación

Existen cuatro tipos de componentes, cada uno de los cuales sirve para un propósito específico y poseen un ciclo de vida propio que define cuando es creado y destruido.

- **Activities**

Es simplemente una pantalla que muestra una interfaz de usuario, dependiendo de la aplicación, por ejemplo si se trata de una aplicación cliente de Twitter, una *Activity* puede ser aquella que muestra el *Timeline* del usuario. De esta manera una aplicación puede estar compuesta de varias *activities* que en conjunto forman la aplicación.

- **Servicios**

Es un componente que corre en segundo plano y realizar pues realiza operaciones que toman un tiempo considerable. Los servicios no proveen una interfaz para el usuario, por ejemplo un servicio puede reproducir música mientras el usuario se encuentra en una aplicación diferente.

- **Proveedor de contenidos (*Content Provider*)**

Administra un conjunto compartido de datos de una aplicación, se puede almacenar datos en el sistema de archivos, una base de datos o la web, cualquier lugar al que la aplicación tenga acceso. A través de un proveedor de contenidos otras aplicaciones pueden consultar o incluso modificar los datos en caso de que sea permitido. Un ejemplo de un proveedor de contenidos es



aquel que administra la información de los contactos almacenados en el dispositivo.

- **Broadcast receivers**

Es un componente que responde a los anuncios del sistema como por ejemplo cuando la pantalla se apaga, existe poca carga en la batería, etc. No poseen una interfaz de usuario como las *activities* pero usan notificaciones que permite alertar al usuario cuando un evento de estas características ha ocurrido.

La estructura de programación que se usa para activar cada uno de estos componentes en una aplicación, se la puede encontrar detallada en la página oficial de desarrolladores Android en donde además existe varios ejemplos para comprender su funcionamiento, el objetivo de la presente es brindar una idea de cómo trabajan las aplicaciones en este sistema operativo.

5.1.3. Recursos de una aplicación

Las aplicaciones desarrolladas para Android además del código, están compuestas de otros recursos como imágenes, archivos de audio o cualquier otro elemento que forme parte de la interfaz del usuario. Para definir menús, estilos, colores, animaciones, etc. Se hace uso de archivos XML debido a que permiten modificar o actualizar características de la App sin necesidad de modificar el código del programa.

Cada recurso que sea incluido en un proyecto de desarrollo de una aplicación, tendrá su propio identificador el cual es usado para hacer referencia al mismo desde el código de la App o desde otros recursos definidos en XML.

5.2. ANDROID DEVELOPER TOOLS (ADT)

Para desarrollar aplicaciones Android, se puede hacer uso de dos IDEs disponibles, Android Studio y Eclipse, éste último mediante el uso del *plugin* ADT el cual provee un conjunto de herramientas que se integran al mismo. Aunque el IDE oficial de desarrollo es Android Studio, esto fue anunciado hace muy poco tiempo. Desde la aparición del sistema operativo Android y la disponibilidad del mismo para que los desarrolladores puedan crear aplicaciones, se hizo uso del ADT mediante Eclipse, por esta razón la documentación y recursos disponibles en la web como tutoriales, blogs o video tutoriales para desarrollar aplicaciones son mayoritariamente enfocados a éste IDE.

La interfaz de Eclipse ofrece: editor de código, editor de Layouts y soporte de reconstrucción de Layouts.

5.3. DESCRIPCIÓN GENERAL DE FUNCIONAMIENTO DE LA APLICACIÓN

La aplicación desarrollada permite al usuario utilizar la misma como una llave electrónica adicional a la tarjeta magnética asignada. Para ello el usuario debe ingresar las credenciales respectivas en la aplicación de forma que se autentique con el sistema, por esta razón la base de datos fue diseñada con un campo *password* o contraseña para los profesores debido a que la misma en conjunto con el nombre de usuario (número de cédula) son necesarias para utilizar la aplicación.

Una vez ingresado el usuario y la contraseña, la aplicación se comunica con una API desarrollada en PHP la cual está ubicada en el servidor y se encarga de recibir las credenciales del usuario para compararlas con las disponibles en la base de datos, en caso de coincidir se envía una respuesta al dispositivo la cual notifica la correcta autenticación permitiendo entonces el acceso a las funciones de la aplicación.

Después de haber ingresado, el usuario podrá seleccionar el aula o laboratorio y obtener acceso al mismo. Para lograrlo nuevamente la aplicación hace uso de la API la cual recibirá el aula seleccionada, obtendrá de la base de datos el ID del dispositivo prototipo y Token de acceso del mismo para por último conectarse al mismo a través de la API de Spark y de forma segura (HTTPS) habilitar el acceso. La siguiente figura describe gráficamente las conexiones que se realizan en el proceso.

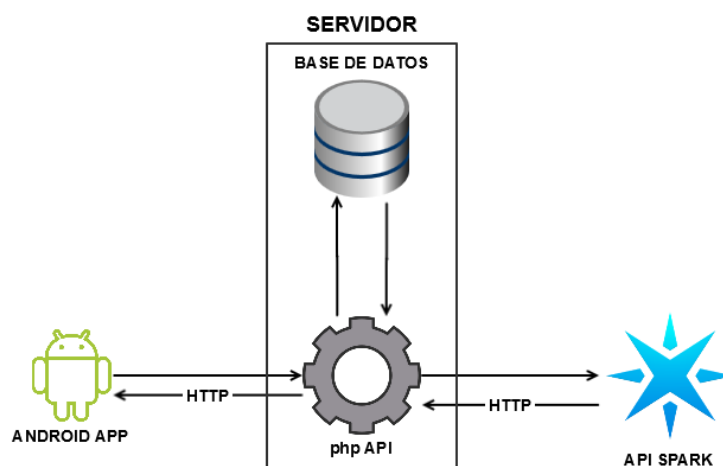


FIGURA 52 CONEXIONES ENTRE APP, API Y API DE SPARK. FUENTE: EL AUTOR

5.4. DISEÑO E IMPLEMENTACIÓN

La interfaz diseñada para la aplicación consta de una Activity para el Login o ingreso de credenciales. Una vez el usuario ha sido autenticado, otra Activity muestra en una lista desplegable las aulas disponibles, se muestra al usuario.



Una vez que el usuario ingresa las credenciales, estas deben ser comparadas en la base de datos del servidor en la que se encuentran almacenados los datos de los usuarios. Para hacerlo la aplicación, mediante una conexión HTTP, envía los datos al servidor en donde la función adecuada de la API los procesará.

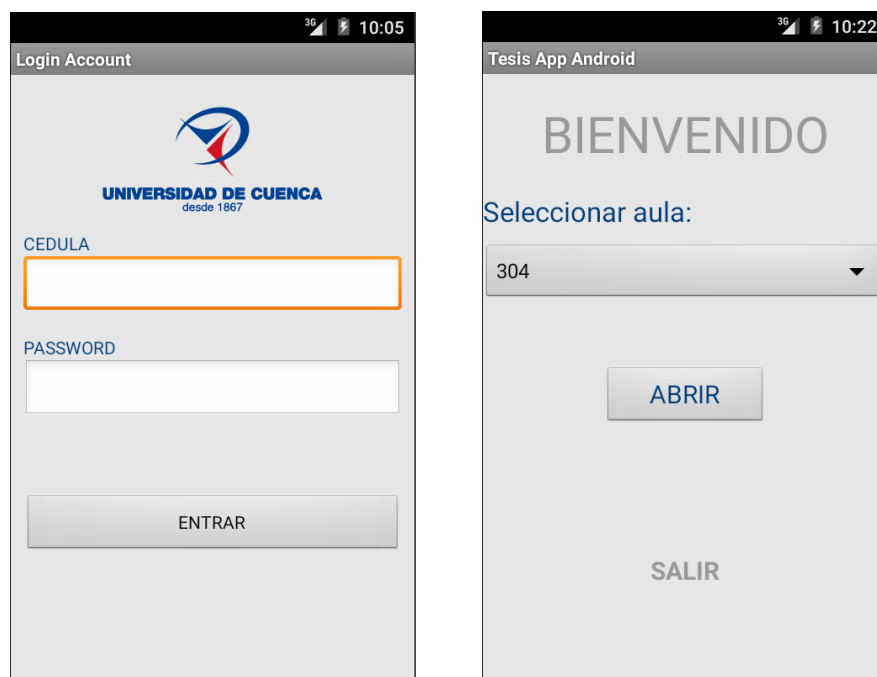


FIGURA 53 PANTALLAS DE INGRESO Y SELECCIÓN DE AULA. FUENTE: EL AUTOR

El mensaje
en formato
como sigue:

```
{
  "tag": "login",
  "cedula": "0104741723",
  "password": "0104741723"
}
```

enviado está
JSON y es

Se hace uso de una etiqueta (tag) para indicar a la API la acción que se está solicitando. En caso de que el usuario haya ingresado correctamente sus credenciales, el servidor enviará una respuesta a la App, de igual manera en formato JSON. Para visualizar la respuesta se hace uso del software POSTMAN, el cual permite enviar solicitudes HTTP (no solo GET).

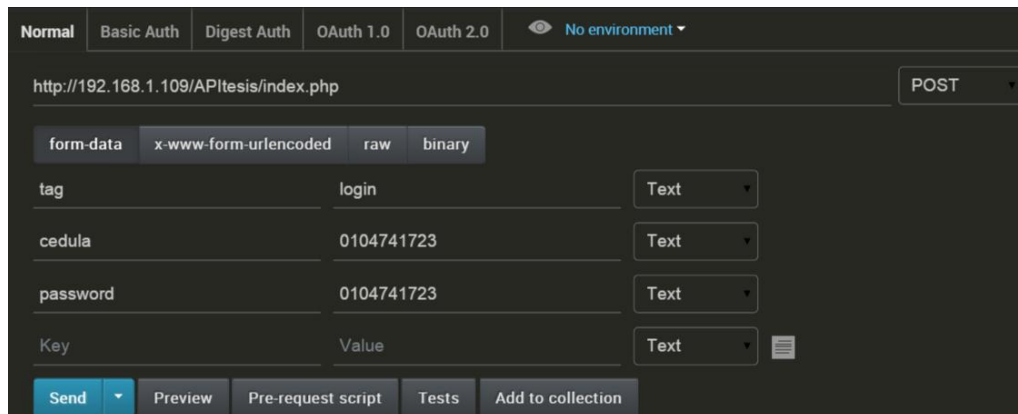


FIGURA 54 SOLICITUD POST PARA LOGIN DEL USUARIO ENVIADA DESDE POSTMAN. FUENTE: EL AUTOR

Como se puede apreciar en la figura, se realiza una solicitud tipo POST al servidor 192.168.1.109 en el cual se encuentra la API. La respuesta enviada desde el servidor se la puede observar en la siguiente figura.

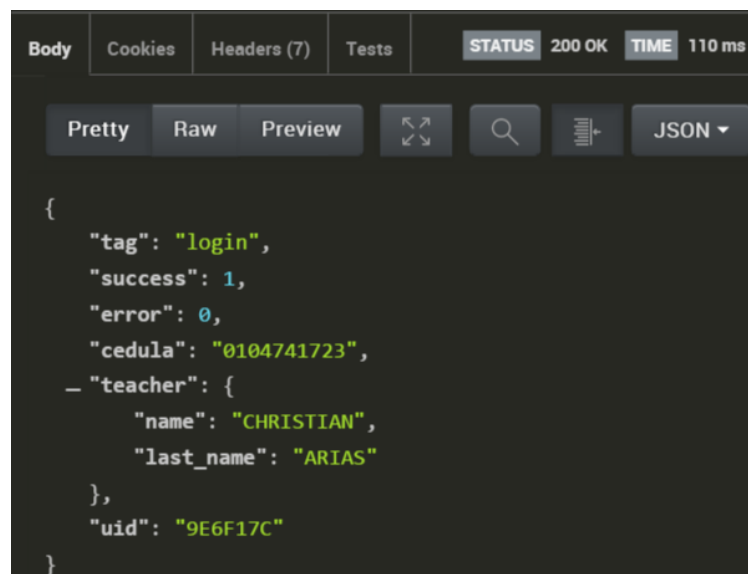


FIGURA 55 RESPUESTA JSON ENVIADA DESDE EL SERVIDOR A LA APP. FUENTE: EL AUTOR

La respuesta que se muestra en la figura consta de la etiqueta *login* que es la acción que se solicitó al servidor, en este caso las credenciales son correctas y se notifica con el valor *success* (éxito) en 1, en caso contrario si las credenciales son incorrectas el valor activado a 1 será error. Las credenciales del usuario acompañan el mensaje.



Esta es la forma en que la aplicación y el servidor se comunican. En caso de que la respuesta sea exitosa, la aplicación muestra el *dashboard* caso contrario se muestra un mensaje indicando que las credenciales son incorrectas.

Una vez el usuario se ha autenticado, la aplicación solicita de la misma manera que antes, el listado de aulas que tienen instalado el prototipo. De esta forma el listado no es estáticamente almacenado en la aplicación sino se la consulta al servidor cada vez que se ingrese a la misma, así en caso de que se agreguen nuevos prototipos no será necesario cambiar código en la aplicación.

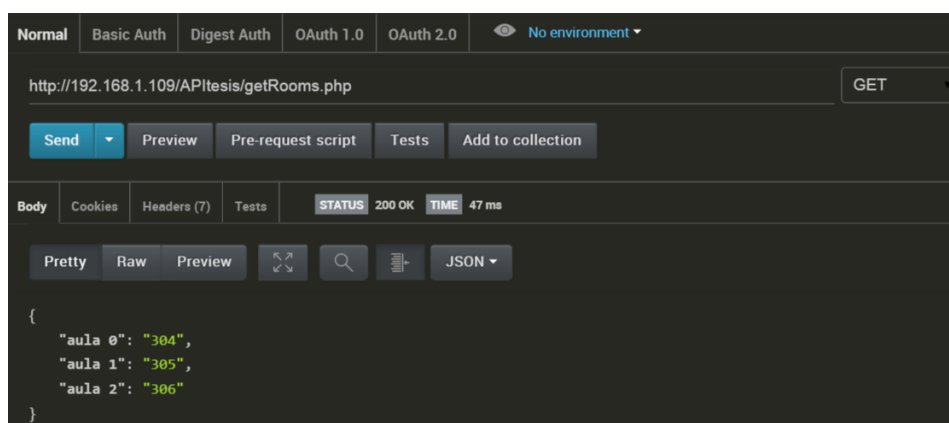


FIGURA 56 LISTADO DE AULAS EN FORMATO JSON, RESPUESTA ENVIADA DESDE EL SERVIDOR. FUENTE: EL AUTOR

La figura anterior muestra el listado de aulas en formato JSON, en este caso la solicitud que se realizó fue de tipo GET ya que no se consideró necesario utilizar POST al no enviar ningún tipo de dato al servidor sino más bien se está solicitando un recurso al mismo.

Una vez el listado ha sido recibido, la aplicación muestra al usuario la misma de la siguiente manera:

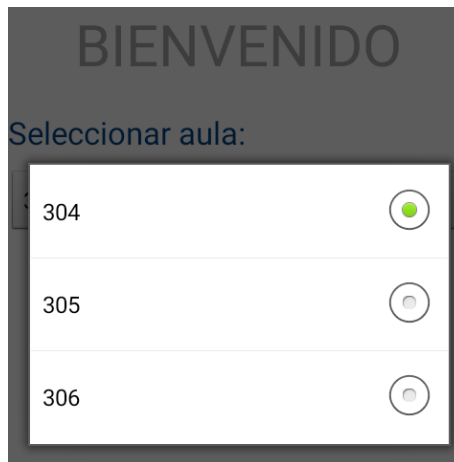


FIGURA 57 LISTADO DE AULAS DISPONIBLES. FUENTE: EL AUTOR

Seleccionada el aula, solo resta abrir la misma. Cuando el usuario selecciona la opción abrir nuevamente la aplicación se comunica con el servidor enviando un mensaje en formato JSON como sigue:

```
{
  "tag": "open",
  "room": "305",
  "cedula": "0104741723"
}
```

Después de comprobar que el usuario se encuentre autenticado con el sistema, la API obtiene los datos del prototipo, ID y Token de acceso para llamar a la función respectiva mediante la API de Spark. Nuevamente se notifica a la aplicación con una respuesta JSON indicando si la solicitud fue realizada con éxito.

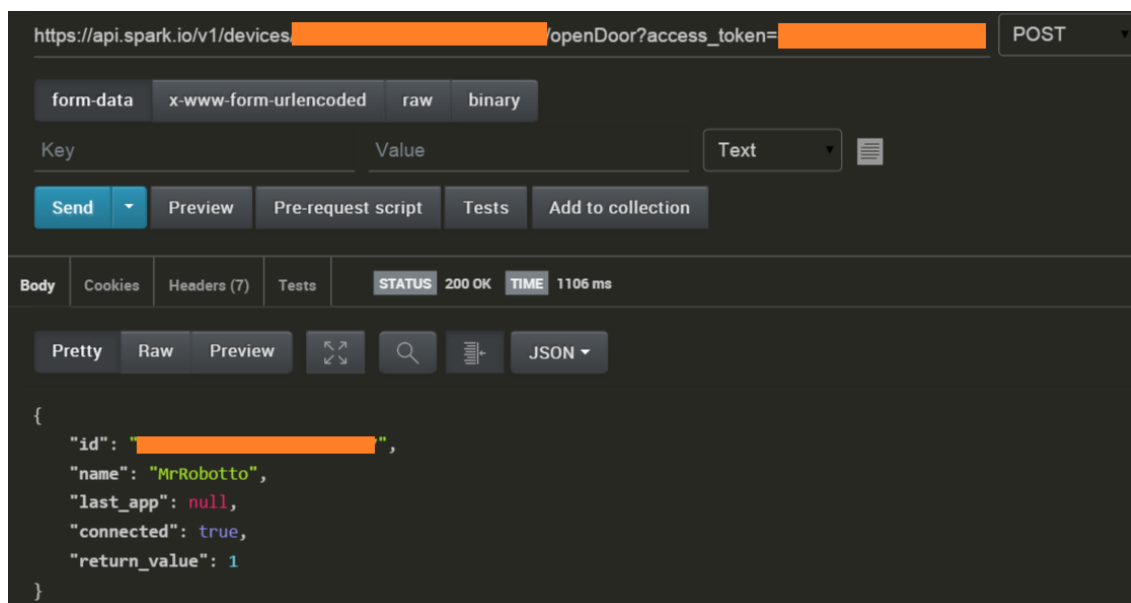


FIGURA 58 RESPUESTA ENVIADA POR LA API DE SPARK. FUENTE: EL AUTOR

En la figura anterior se indica mediante el valor `return_value` que se logró abrir la puerta lo cual es el indicador para que la API del servidor también pueda responder confirmando la apertura a la aplicación.

```
{
  "tag": "open",
  "success": 1,
  "error_id": 0,
  "error_token": 0,
  "error_curl": 0
}
```

FIGURA 59 RESPUESTA ENVIADA DESDE EL SERVIDOR A LA APP. FUENTE: EL AUTOR

De esta manera es como la aplicación en conjunto con las API del servidor y de Spark, permiten la apertura de la puerta de una aula o laboratorio desde un Smartphone o Tablet que trabaja con el sistema operativo Android.



CONCLUSIONES Y RECOMENDACIONES

La presente tesis se planteó pensando en aplicar algunos de los conocimientos adquiridos a lo largo de carrera y no solo crear un documento que recopile información sobre algún tema en particular.

Se ha diseñado e implementado un dispositivo con el objetivo de que en algún momento, pueda ser utilizado para gestionar el acceso a las aulas o laboratorios por parte de los docentes, optimizando recursos humanos y tiempo entre las diferentes sesiones de cátedra.

El dispositivo desarrollado está inspirado e influenciado directamente por la tendencia actual denominada Internet de las cosas, en la que todo dispositivo genera información, puede ser monitoreado y controlado desde cualquier lugar del mundo, basta con tener una conexión a Internet y una computadora, teléfono inteligente o Tablet.

Actualmente varios dispositivos similares han sido presentados, con la diferencia de que el mercado objetivo de los mismos es el hogar. La implementación realizada en esta tesis a diferencia de los dispositivos disponibles en el mercado, está enfocada y desarrollada específicamente para el uso en la Facultad. Finalmente considerando que los dispositivos disponibles en el mercado han sido desarrollados por empresas extranjeras con personal dedicado a esta tarea, la presente tesis ha sido un reto personal al decidir no solo desarrollar el prototipo electrónico sino también el software que se conecta y permite interactuar y administrar el mismo.

En cuanto al diseño para gestionar y comunicarse con el prototipo creado, se garantiza que es completamente escalable, permitiendo desplegar la cantidad necesaria de prototipos en caso de ser implementado en el algún momento, además de que los requerimientos para el software de servidor son mínimos pudiendo ser instalado en cualquier computadora.

La selección de la plataforma de desarrollo electrónico ha sido de gran ayuda para el proyecto debido a que viene acompañada de un lenguaje de programación simple y librerías que disminuyen la dificultad de tareas como la interacción con el módulo Wi-Fi o la creación de sockets TCP/IP, lo cual permitió centrar esfuerzos en el funcionamiento específico del prototipo.

Como recomendación en caso de que se desee implementar el prototipo presentado, fuese interesante desarrollar la integración con el servidor LDAP de manera que se pueda realizar la verificación del usuario utilizando la autenticación ya existente.



Por otra parte, al tener disponible en la plataforma de desarrollo Spark Core varios pines analógicos y digitales de entrada y salida, se podría considerar funcionalidades adicionales dentro del programa para interactuar con sensores de cualquier tipo que puedan ser conectados, como por ejemplo: sensores de movimiento para determinar si existe alguien dentro de un aula o sensores que determinen si la puerta está abierta o cerrada. En definitiva al tener disponibles varios pines, las posibilidades de utilizarlos con cualquier sensor permitiría darle funcionalidades adicionales al prototipo.



REFERENCIAS

- [1] RFID and Sensor Networks: Architectures, Protocols, Security, and Integrations, Yan Zhang, Laurence T. Yang, Jiming Chen
- [2] RFID Handbook: Fundamentals and Applications in Contactless Smart Cards, Radio Frequency Identification and Near-Field Communication, Klaus Finkenzeller, Tercera Edición, Wiley And Sons 2010
- [3] The Internet of Things: From RFID to the Next Generation Pervasive Networked Systems, Lu Yan, Yan Zhang, Laurence T. Yang, Huansheng Ning, Auerbach Publications 2008
- [4] <https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi>
- [5] Arquitectura ARM: <http://www.arm.com/products/processors/instruction-setarchitectures/index.php>
- [6] The Web Application Hackers Handbook 2nd Edition, Dafydd Stuttard, Marcus Pinto, Wiley Publishing 2011.
- [7] TCP/IP Tutorial and Technical Overview, Parziale Lydia, Britt David, IBM 2006-
- [8] http://www.ecured.cu/index.php/Arquitectura_Cliente_Servidor
- [9] <http://www.cs.tut.fi/kurssit/ELT-53206/lecture01.pdf>
- [10] <http://www.mcgraw-hill.es/bcv/guide/capitulo/8448148797.pdf>
- [11] <http://oscarstorrío.com/post/2010/10/12/Transformacion-del-Modelo-ER-al-Modelo-Relacional.aspx>
- [12] http://www.wikiwand.com/es/Modelo_entidad-relaci%C3%B3n
- [13] http://en.wikibooks.org/wiki/Structured_Query_Language/Language_Elements
- [14] <http://dev.mysql.com/doc/refman/5.6/en/what-is-mysql.html>
- [15] <http://www.jorgesanchez.net/bd/bdrelacional.pdf>
- [16] www.jorgesanchez.net/bd/bdrelacional.pdf
- [17] <http://dev.mysql.com/doc/refman/5.6/en/introduction.html>
- [18] http://www.diablotin.com/librairie/networking/firewall/appc_12.htm
- [19] <http://php.net/manual/en/intro-what-is.php>
- [20] <http://www.wikiwand.com/es/HTML>
- [21] http://www.wikiwand.com/en/Scripting_language



- [22] <http://www.definicionabc.com/tecnologia/php.php>
- [23] http://www.wikiwand.com/en/Software_framework
- [24] <http://book.cakephp.org/1.3/en/The-Manual/Beginning-With-CakePHP/What-is-CakePHP-Why-Use-it.html>
- [25] <http://michelletorres.mx/mvc-y-su-importancia-en-la-web/#.VMcPWY6rPLV>
- [26] <http://book.cakephp.org/2.0/es/cakephp-overview/understanding-model-view-controller.html>
- [27] <http://book.cakephp.org/2.0/en/core-libraries/components/authentication.html>
- [28] RC522 Datasheet
- [29] <http://www.pudn.com/downloads98/sourcecode/embed/detail401401.html>
- [30] <https://code.google.com/p/mfrclib4arm/>
- [31] <https://books.google.com.ec/books?id=hwNWRHc79PMC>
- [32] http://www.wikiwand.com/es/Arquitectura_ARM



ANEXO: CÓDIGOS DE PROGRAMACIÓN

A1.1. Programa del prototipo

Spark Dev - Arduino

rfid.ino

```

/*****
Title:          Prototipo para gestión de acceso
Autor:          Christian Xavier Arias Ordóñez
Licencia:       Creative Commons (BY-NC)
                Reconocimiento al autor original: Christian Arias
                Ordóñez
                No comercial.
                Se puede copiar, modificar y distribuir sin fines
                comerciales.
Fecha inicial:  18/08/2014
Última modificación: 20/03/2015
Versión:        2.5
Disponible en:  https://github.com/christian-arias
*****/

#include "MFRC522.h"
#define SS_PIN SS
#define RST_PIN D2

//Variables utilizadas para crear el Cliente TCP
TCPCClient cliente;
IPAddress server(192, 168, 1, 114);
uint16_t port = 80;

boolean estado;
boolean ledStatus = false;
byte id[4];
int led1 = D7;
int pulso = D1;
int buzzer = D0;
char UID[4];
String myID = "53ff75065075535140551487";

MFRC522 mfrc522(SS_PIN, RST_PIN);          // Crear instancia del MFRC522

//=====//
//                               FUNCIONES                               //
//=====//
void sonido(){
    digitalWrite(buzzer, HIGH);
    delay(100);
    digitalWrite(buzzer, LOW);
}

boolean verificarConexion(){
    if (cliente.connected()){

```



```

        estado = true;
    } else {
        estado = false;
    }
    return estado;
}

void sendUID_To_PC(){
    Serial.print("Numero de Bytes: ");
    Serial.print(mfrc522.uid.size);
    Serial.println();
    Serial.print("UID:");

    for (byte i = 0; i < mfrc522.uid.size; i++) {
        Serial.print(mfrc522.uid.uidByte[i], HEX);

        //Almacenar el codigo en una variable para enviarlo al servicio Web
        id[i] = (mfrc522.uid.uidByte[i]);
    }

    byte piccType = mfrc522.PICC_GetType(mfrc522.uid.sak);
    Serial.println(" Tipo PICC: ");
    Serial.println(mfrc522.PICC_GetTypeName(piccType));
    if (
        piccType != MFRC522::PICC_TYPE_MIFARE_MINI
        && piccType != MFRC522::PICC_TYPE_MIFARE_1K
        && piccType != MFRC522::PICC_TYPE_MIFARE_4K) {
        return;
    }
}

void saveUID(){
    String idString1 = String (id[0], HEX);
    String idString2 = String (id[1], HEX);
    String idString3 = String (id[2], HEX);
    String idString4 = String (id[3], HEX);
    //UID = idString1+idString2+idString3+idString4;
    sprintf(UID, "%x%x%x%x", id[0], id[1], id[2], id[3]);
}

void openSocketTCP(){
    cliente.connect(server, port);
    String m= "Conexión Exitosa a Servidor ";
    String oct1 = String(server[0]);
    String oct2 = String(server[1]);
    String oct3 = String(server[2]);
    String oct4 = String(server[3]);
    String servidor = String(oct1 + "." + oct2 + "." + oct3 + "." + oct4);
    if (cliente.connected()){
        Serial.println(m + servidor);
        Serial.println(WiFi.localIP());
    } else {
        Serial.println("Falló conexión!");
        Serial.println(WiFi.localIP());
    }
}

void sendToAPIServer(){
    cliente.flush();
    delay(1);
    cliente.println("GET /APITesis/receiveUID.php?myID=" + myID + " HTTP/1.0");
}

```



```

Serial.println(myID);
cliente.println("Host: 192.168.1.114");
cliente.println("Connection: close");
cliente.println();
delay(10);
cliente.flush();
delay(10);
cliente.stop();
}

//=====//
//      FUNCIONES QUE SE PUEDEN LLAMAR MEDIANTE LA REST API DE SPARK      //
//=====//

int openDoor(String command)
{
    digitalWrite(pulso,1);
    delay(20);
    digitalWrite(pulso,0);
    ledStatus = !ledStatus;
    if (ledStatus) digitalWrite(led1, 1);
    else digitalWrite(led1, 0);
    return 1;
}

//=====//
//                        CONFIGURACIONES                                //
//=====//

void setup() {
    Serial.begin(9600);
    delay(2000); // Inicializar comunicacion Serial
    //while(!Serial.available()) SPARK_WLAN_Loop(); //Bucle infinito hasta que se
    //presione una tecla en una conexion serial
    Serial.println("
=====");
    Serial.println("|  PROTOTIPO  ELECTRONICO:  GESTION  DE  ACCESO  A  AULAS  Y
LABORATORIOS UCUENCA |");
    Serial.println("
=====");

    /**Publicando funciones y variables para ser llamadas mediante la API de
    SPARK**/
    Spark.function("openDoor", openDoor);
    Spark.variable("uid", UID, STRING);
    //*****//

    // Configuracion de pines
    pinMode(led1, OUTPUT);    //Pin para LED
    pinMode(pulso, OUTPUT);   //Pin para pulso puerta
    pinMode(buzzer, OUTPUT);  //Pin para pulso puerta

    //Inicializacion
    digitalWrite(led1, LOW);   //Apago LED
    digitalWrite(pulso, LOW);  //Apago Buzzer
    digitalWrite(buzzer, LOW); //Apago Buzzer
    SPI.begin();               // Inicializar comunicacion SPI
    SPI.setClockDivider(SPI_CLOCK_DIV8);
    mfr522.PCD_Init();         // Inicializar el modulo MFRC52211

```



```
}

//=====//
//                                LOOP                                //
//=====//

void loop() {
    MFRC522::MIFARE_Key key;
    for (byte i = 0; i < 6; i++) {
        key.keyByte[i] = 0xFF;
    }
    // BUSCAR TARJETAS
    if ( ! mfrc522.PICC_IsNewCardPresent()) {
        return;
    }

    // SELECCIONAR LA TARJETA ENCONTRADA
    if ( ! mfrc522.PICC_ReadCardSerial()) {
        return;
    }

    sonido();
    saveUID(); //Guardar el UID de la tarjeta, el cual será solicitado mediante la
nube (SparkOS) por parte de la API(PHP) del sistema

    //*****Este bloque solo envia a la PC datos en forma serial*****//
    sendUID_To_PC();
    //*****//

    //ABIR UN SOCKET TCP DE CONEXION AL SERVIDOR HTTP, LUEGO ENVIAR LOS DATOS
NECESARIOS AL SERVICIO WEB ADECUADO//
    openSocketTCP();
    if (verificarConexion()){
        sendToAPIServer();
    }
    delay(1500);
}
```



A1.2. Aplicación Web

Cake PHP

VISTAS

login.ctp

```
<h2>SISTEMA DE GESTION DE ACCESO A LAS AULAS DE LA FACULTAD DE INGENIERIA</h2>

<?php
    echo $this->Session->flash('auth');
    echo $this->Form->create('User');
    echo $this->Form->input('username', array('label'=>'Usuario'));
    echo $this->Form->input('password', array('label'=>'Contraseña'));
    echo $this->Form->end('INGRESAR');
?>
```

Index.ctp (Administrador)

```
<P ALIGN="right">
    <?php
        echo "BIENVENIDO: " . $this->Session->read('Auth.User.name') . " " . $this->Session->read('Auth.User.last_name') .
            " " . $this->Html->link('Salir', array('action'=>'logout'));
    ?>
</P>

<?php
    echo $this->Html->link('Administrar Profesores',
        array(
            'controller' => 'teachers',
            'action' => 'index'
        )
    );

    echo "&nbsp; <br> <br>";

    echo $this->Html->link('Administrar Prototipos',
        array(
            'controller' => 'devices',
            'action' => 'index'
        )
    );

    echo "&nbsp; <br> <br>";

    echo $this->Html->link('Administrar Aulas',
        array(
            'controller' => 'rooms',
            'action' => 'index'
        )
    );
?>
```




index.ctp (Profesores)

```

<P ALIGN="right">
    <?php
        echo "BIENVENIDO: " . $this->Session->read('Auth.User.name') . " " .
        $this->Session->read('Auth.User.last_name') .
        " " . $this->Html->link('Salir', array('action'=>'logout'));
    ?>
</P>

<h2>Profesores</h2>

<?php
    echo $this->Html->link('Agregar Profesor', array('action'=>'add'));
    echo "<br> <br>";
?>

<table>
    <tr>
        <th><?php echo $this->Paginator->sort('id', 'Cedula')?></th>
        <th><?php echo $this->Paginator->sort('name', 'Nombre')?></th>
        <th><?php echo $this->Paginator->sort('last_name', 'Apellido')?></th>
        <th>Acciones</th>
    </tr>
    <?php foreach($profesores as $k=>$profesor){?>
        <tr>
            <td><?php echo h($profesor['Teacher']['id']); ?></td>
            <td><?php echo h($profesor['Teacher']['name']); ?></td>
            <td><?php echo h($profesor['Teacher']['last_name']); ?></td>
            <td>
                <?php
                    //Uso el helper Html
                    echo $this->Html->link('Editar',
                        array('action'=>'edit', $profesor['Teacher']['id']));
                ?>

                &nbsp;

                <?php
                    echo $this->Form->postLink('Eliminar',
                        array('action'=>'delete',
                            $profesor['Teacher']['id'],
                            array('confirm'=>'Realmente desea eliminar a '.
                                $profesor['Teacher']['name'] . " " . $profesor['Teacher']['last_name'] . ' ?')));
                ?>
            </td>
        </tr>
    <?php } ?>
</table>

```



add.ctp

```
<h2>Agregar Profesor</h2>

<?php
    echo $this->Form->create('Teacher');

    echo $this->Form->input('id', array('type'=>'text',
'maxlength'=>'10', 'label'=>'Cedula'));
    echo $this->Form->input('name', array('label'=>'Nombre'));
    echo $this->Form->input('last_name', array('label'=>'Apellido'));
    echo $this->Form->input('uid', array('label'=>'UID'));
    echo $this->Form->input('password', array('label'=>'Password'));
    echo $this->Form->end('GUARDAR');
    echo $this->Html->link('CANCELAR', array('action'=>'goBack'));

?>
```

edit.ctp

```
<h1>Editar profesor</h1>
<?php
    echo $this->Form->create('Teacher', array('action'=>'edit'));
    echo $this->Form->input('id', array('type'=>'hidden'));
    echo $this->Form->input('name', array('label'=>'Nombre', ));
    echo $this->Form->input('last_name', array('label'=>'Apellido'));
    echo $this->Form->input('uid', array('label'=>'UID'));
    echo $this->Form->input('password', array('label'=>'Password'));
    echo $this->Form->end('Guardar profesor');

?>
```

MODELO

Teacher.php

```
<?php
class Teacher extends AppModel {
    var $validate = array(
        'id' => array('allowEmpty' => false),
        'name' => array('allowEmpty' => false),
        'last_name' => array('allowEmpty' => false),
        'uid' => array('allowEmpty' => false),
        'password' => array('allowEmpty' => false)
    );

    public function beforeSave($options = array()){
        if(isset($this->data[$this->alias]['password']))
            $this->data[$this->alias]['password'] = password_hash($this->data[$this->alias]['password'], PASSWORD_DEFAULT);
        return true;
    }
}

?>
```



CONTROLADOR

TeachersController.php

```
<?php
class TeachersController extends ApplicationController{
    public $helpers = array('html', 'Form')
;    public $components = array('Session');

    public function beforeFilter(){
        if (!$this->Auth->loggedIn()) {
            $this->Auth->authError = false;
        }
    }

    public function goBack(){
        $this->redirect(array('action'=>'index'));
    }

    public function index(){
        $this->set('title_for_layout', 'ADMINISTRAR PROFESORES');
        $this->Teacher->recursive=0;
        $this->set('teachers', $this->paginate());
        $params = array('order' => 'name desc');
        $this->set('profesores', $this->Teacher->find('all',$params));
    }

    public function add(){
        $this->set('title_for_layout', 'AGREGAR PROFESOR');
        $this->Teacher->set($this->request->data);
        if ($this->request->is('post')){
            if($this->Teacher->save($this->request->data)){
                $this->Session->setFlash('Profesor guardado');
                $this->redirect(array('action'=>'index'));
            }
        }
    }

    public function edit($id = null){
        $this->set('title_for_layout', 'EDITAR PROFESOR');
        $this->Teacher->id = $id;
        if($this->request->is('get')){
            $this->request->data = $this->Teacher->read();
        }
        else {
            if($this->Teacher->save($this->request->data)){
                $this->Session->setFlash('Profesor guardado');
                $this->redirect(array('action' => 'index'));
            }
            else {
                $this->Session->setFlash('No se pudo guardar');
            }
        }
    }

    public function delete($id){
        if ($this->request->is('get')){
            throw new MethodNotAllowedException();
        }
    }
}
```



```

    }
    else{
        if($this->Teacher->delete($id)){
            $this->Session->setFlash('Profesor eliminado');
            $this->redirect(array('action'=>'index'));
        }
    }
}

public function logout(){
    if($this->Auth->logout()){
        return $this->redirect($this->Auth->redirect());
    }
}
}

```

UsersController.php

```

<?php
class UsersController extends ApplicationController{
    public $helpers = array('html', 'Form');
    public $components = array('Session');

    public function index(){

    }

    public function login(){
        if ($this->request->is('post')){
            if ($this->Auth->login()){
                return $this->redirect($this->Auth->redirect());
            }
            else{
                $this->Session->setFlash('Usuario o contraseña no
                validos');
            }
        }
    }

    public function logout(){
        if($this->Auth->logout()){
            return $this->redirect($this->Auth->redirect());
        }
    }
}

```

AppController.php

```

<?php
/**
 *
 * CakePHP(tm) : Rapid Development Framework (http://cakephp.org)
 * Copyright (c) Cake Software Foundation, Inc. (http://cakefoundation.org)
 *
 * Licensed under The MIT License
 * For full copyright and license information, please see the LICENSE.txt
 * Redistributions of files must retain the above copyright notice.

```



```

*
*  @copyright      Copyright (c) Cake Software Foundation, Inc.
(http://cakefoundation.org)
*  @link           http://cakephp.org CakePHP(tm) Project
*  @package        app.Controller
*  @since          CakePHP(tm) v 0.2.9
*  @license        http://www.opensource.org/licenses/mit-license.php MIT License
*/

App::uses('Controller', 'Controller');

/**
 * Application Controller
 *
 * Add your application-wide methods in the class below, your controllers
 * will inherit them.
 *
 * @package        app.Controller
 * @link           http://book.cakephp.org/2.0/en/controllers.html#the-app-
controller
 */
class AppController extends Controller {

    public $components = array(
        'Session',
        'Auth' => array(
            'loginRedirect' => array(
                'controller' => 'users',
                'action' => 'index'
            ),
            'logoutRedirect' => array(
                'controller' => 'users',
                'action' => 'login'
            ),
            'authenticate' => array(
                'Form' => array(
                    'passwordHasher' => array(
                        'className' => 'Simple',
                        'hashType' => 'sha1'
                    )
                )
            ),
            'authError'=>'Necesita ingresar sus credenciales para ingresar al
Sistema'
        )
    );
}

```



A1.3. API

PHP

config.php

```
<?php
//Configuracion de variables de la Base de Datos
define("DB_HOST", "localhost");
define("DB_USER", "root");
define("DB_PASSWORD", "");
define("DB_DATABASE", "tesis");
?>
```

DB_Connect.php

```
<?php
class DB_Connect {
    protected $con;
    // Constructor
    function __construct() {
    }

    // Destructor
    function __destruct() {
        // $this->close();
    }

    // Conectar a la BD (MySQL)
    public function connect() {
        require_once 'config.php'; // Requiere el archivo de configuración
        // Conectando a MySQL
        $con = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD); //Las variables
        // Seleccionar la base de datos de la tesis
        mysqli_select_db($con, DB_DATABASE);
        // Regresa en handler de la conexión
        return $con;
    }

    // Cerrar la conexión a la BD
    public function close() {
        mysqli_close();
    }

    public function sendLink(){
        return $con;
    }
}
?>
```

DB_Functions.php



```

<?php
class DB_Functions {
    private $db;

    // Constructor
    function __construct() {
        require_once 'DB_Connect.php';
        // Instancia de la clase DB_Connect
        $this->db = new DB_Connect();
        //Llama al método de conexión a la BD
        $this->db->connect();
    }

    // Destructor
    function __destruct() {
    }

    /**
     * Encontrar profesor mediante cedula y password. Find teacher by cedula (id)
     and password.
     * Básicamente comprobar los datos de autenticación.
     * @param cedula
     * @param password
     */
    public function findByCedulaAndPassword($cedula, $password) {
        $con = $this->db->connect();
        $result = mysqli_query($con, "SELECT * FROM teachers WHERE id = '$cedula'")
or die(mysqli_error());
        // Comprobar que se haya encontrado un solo registro. Check if just one
record was found.
        $num_rows = mysqli_num_rows($result);
        if ($num_rows > 0) {
            $row = mysqli_fetch_row($result);
            $encrypted_password = $row[4];
            if (password_verify($password, $encrypted_password)) {
                // Datos de autenticacion correctos. Authentication data is
correct.
                return $row;
            }
        } else {
            // No se encontro al usuario. Uer not found.
            return false;
        }
    }

    //Comprobar si existe un usuario
    public function isUserExisted($cedula) {
        $result = mysql_query("SELECT id from teachers WHERE id = '$cedula'");
        $num_rows = mysql_num_rows($result);
        if ($num_rows > 0) {
            return true; // Existe usuario. User exists.
        } else {
            return false; // No existe el usuario. Not found.
        }
    }

    /**
     * Encriptar password (contraseña). Encrypt password.
     * @param password
     * Función nativa de PHP desde 5.5.0, no es necesario un valor salt

```



personalizado ya que se genera uno seguro automáticamente.

```

    */
    public function hash_password($password) {
        $encrypted = password_hash($password, PASSWORD_DEFAULT);
        return $encrypted;
    }

    public function getRooms(){
        $con = $this->db->connect();
        $result = mysqli_query($con, "SELECT name from rooms");
        $rows = array();

        $i = 0;

        while($r = mysqli_fetch_assoc($result)) {
            $rows['aula '.$i] = $r['name'];
            $i = $i + 1;
        }
        return $rows;
    }

    public function getDeviceId($room){
        $con = $this->db->connect();
        $result = mysqli_query($con, "SELECT id_device from rooms WHERE name =
'$room'");
        $num_rows = mysqli_num_rows($result);
        if ($num_rows == 1) {
            $row = mysqli_fetch_row($result);
            $device_id = $row[0];
            return $device_id;
        } else {
            // No se encontro esa aula
            return false;
        }
    }

    public function getToken($device_id){
        $con = $this->db->connect();
        $result = mysqli_query($con, "SELECT token from devices WHERE id =
'$device_id'");
        $num_rows = mysqli_num_rows($result);
        if ($num_rows == 1) {
            $row = mysqli_fetch_row($result);
            $token = $row[0];
            return $token;
        } else {
            // No se encontro esa aula
            return false;
        }
    }

    public function execCurl($id, $token){
        $ch = curl_init();
        $curlConfig = array(
            CURLOPT_URL
            "https://api.spark.io/v1/devices/" . $id . "/openDoor?access_token=" . $token,
            CURLOPT_POST => true,
            CURLOPT_RETURNTRANSFER => true,
            CURLOPT_SSL_VERIFYPEER => false,
            CURLOPT_POSTFIELDS => array("params=" => "")
        );
    }

```




```

    );

    curl_setopt_array($ch, $curlConfig);
    $result = curl_exec($ch);
    curl_close($ch);

    $json_data = json_decode($result, true);
    $status = strtoupper($json_data['return_value']);

    return $status;
}
}
?>

```

receiveUID.php

```

<?php
//CONEXION A LA BASE DE DATOS MYSQL//
$con = mysqli_connect("localhost", "root", "");
if (!$con) {
    exit('Connect Error (' . mysqli_connect_errno() . ') '
        . mysqli_connect_error());
}

mysqli_set_charset($con, 'utf-8');

mysqli_select_db($con, "tesis");

//Obtengo el ID del dispositivo enviado desde el Spark Core mediante el método
HTTP -> GET
$id = mysqli_real_escape_string($con, htmlentities($_GET["myID"]));

//Realizo la consulta a la base de datos, busco el código enviado en la tabla
teachers
$devices = mysqli_query($con, "SELECT * FROM devices WHERE id='" . $id . "'");
$numeros = mysqli_num_rows($devices);
//echo nl2br("Coincidencias dispositivos: " . $numeros . "\n");

//Compruebo que se haya encontrado solo un registro
if ($numeros == 1) {
    //Obtengo el Token de acceso
    $row = mysqli_fetch_row($devices);
    $token = $row[1];
    //Obtengo el UID almacenado en el Spark Core, el cual corresponde a la última
    tarjeta acercada al sensor RFID
    header('Content-Type: application/json');
    $respuesta =
file_get_contents("https://api.spark.io/v1/devices/" . $id . "/uid?access_token=" . $token)
;
    //echo $respuesta;

    //Muestra la respuesta en formato JSON::Solo para pruebas desde el navegador
    $json_data = json_decode($respuesta, true);
    $uid = strtoupper($json_data['result']);

    //Compruebo que el UID exista en la base de datos de profesores
    $teachers = mysqli_query($con, "SELECT * FROM teachers WHERE uid='" . $uid .
    "'");
    $numeros = mysqli_num_rows($teachers);
}
}
?>

```



```
//echo nl2br("Coincidencias profesores: " . $numero . "\n");

if ($numero == 1){
    $ch = curl_init();
    $curlConfig = array(
        CURLOPT_URL
        "https://api.spark.io/v1/devices/" . $id . "/openDoor?access_token=" . $token,
        CURLOPT_POST => true,
        CURLOPT_RETURNTRANSFER => true,
        CURLOPT_SSL_VERIFYPEER => false,
        CURLOPT_POSTFIELDS => array("params=" => "")
    );

    curl_setopt_array($ch, $curlConfig);
    $result = curl_exec($ch);
    curl_close($ch);
    echo $result;
}

mysqli_free_result($devices);
mysqli_free_result($teachers);
?>
```

getRooms.php

```
<?php
    require_once 'DB_Functions.php';
    $db = new DB_Functions();

    $response = $db->getRooms();
    echo json_encode($response);
?>
```

index.php

```
<?php
/**
 * Maneja las solicitudes a la API enviadas desde la app de Android.
 */

define("id", 0);
define("nombre", 1);
define("apellido", 2);
define("uid", 3);

if (isset($_POST['tag']) && $_POST['tag'] != '') {
    // get tag
    $tag = $_POST['tag'];

    // include db handler
    require_once 'DB_Functions.php';
    $db = new DB_Functions();

    // response Array
    $response = array("tag" => $tag, "success" => 0, "error" => 0);

    // check for tag type
```



```

if ($tag == 'login') {
    // Request type is check Login
    $cedula = $_POST['cedula'];
    $password = $_POST['password'];

    // check for teacher
    $teacher = $db->findByCedulaAndPassword($cedula, $password);
    if ($teacher != false) {
        // Usuario -> Profesor encontrado
        // Echo JSON with success = 1
        $response["success"] = 1;
        $response["cedula"] = $teacher[id];
        $response["teacher"]["name"] = $teacher[nombre];
        $response["teacher"]["last_name"] = $teacher[apellido];
        $response["uid"] = $teacher[uid];
        echo json_encode($response);
    } else {
        // Profesor no encontrado.
        // Echo JSON with error = 1
        $response["error"] = 1;
        $response["error_msg"] = "Verificar cedula o password.";
        echo json_encode($response);
    }
} elseif($tag == 'open'){
    $response = array("tag"=>$tag, "success"=>0, "error_id"=>0, "error_token"=>
0, "error_curl" => 0);
    $room = $_POST['room'];

    //Query SQL id_device
    $device_id = $db->getDeviceId($room);

    if($device_id != false){
        //Query SQL token
        $token = $db->getToken($device_id);

        if ($token == false){
            $response ["error_token"] = 1;
        } else {
            $status = $db->execCurl($device_id, $token);
            if ($status == 1){
                $response ["success"] = 1;
            } elseif ($status == 1) {
                $response ["error_curl"] = 1;
            }
        }
    }

    } else {
        $response ["error_id"] = 1;
    }
    echo json_encode($response);

} else {
    echo "Solicitud no válida. Invalid request!!!";
}
} else {
    echo "<h1>Acceso negado. Denied access.</h1>";
}
}

```



```
?>
```

idCore.php

```
<?php
//CONEXION A LA BASDE DE DATOS MYSQL//
$con = mysqli_connect("localhost", "root", "");
if (!$con) {
    exit('Connect Error (' . mysqli_connect_errno() . ') '
        . mysqli_connect_error());
}

mysqli_set_charset($con, 'utf-8');

mysqli_select_db($con, "tesis");

//Obtengo el código enviado desde el Spark Core mediante el método HTTP -> GET
$code = mysqli_real_escape_string($con, htmlentities($_GET["code"]));

//Realizo la consulta a la base de datos, busco el codigo enviado en la tabla
teachers
$teachers = mysqli_query($con, "SELECT * FROM teachers WHERE code='" . $code .
    "'");
echo "aqui>>>";
$numeros = mysqli_num_rows($teachers);
echo $numeros;

//Compruebo que se haya encontrado solo un registro
if (mysqli_num_rows($teachers) == 1) {
    //$respuesta
file_get_contents("https://api.spark.io/v1/devices/53ff75065075535140551487/led -d
access_token=fc592790678209a25b7ea2e6d70168f09ddb6bde -d params=l7,HIGH");
    //echo $respuesta;

    $myDevice = "53ff75065075535140551487"; //id del
dispositivo
    $myToken = "4833e9bd434662d7e8297659947263508040cb8c"; //Token de acceso

    $ch = curl_init();
    $curlConfig = array(
        CURLOPT_URL => "https://api.spark.io/v1/devices/" . $myDevice .
"/ledws?access_token=" . $myToken,
        CURLOPT_POST => true,
        CURLOPT_RETURNTRANSFER => true,
        CURLOPT_SSL_VERIFYPEER => false,
        CURLOPT_POSTFIELDS => array("params=" => "")
    );

    curl_setopt_array($ch, $curlConfig);
    $result = curl_exec($ch);
    curl_close($ch);
    echo $result;
}
mysqli_free_result($teachers);
?>
```



A1.4. Aplicación Móvil

Eclipse ADT – Java

DashboardActivity.java

```
package org.app.tesis.activities;

import java.util.ArrayList;
import java.util.List;

import org.app.tesis.R;
import org.app.tesis.libraries.UserFunctions;
import org.json.JSONException;
import org.json.JSONObject;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.Gravity;
import android.view.View;
import android.widget.AdapterView;
import android.widget.Button;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.Toast;

public class DashboardActivity extends Activity {
    UserFunctions userFunctions;
    Button btnLogout;
    Spinner spinnerRooms;
    Button btnOpen;
    TextView txtSat;

    //JSON nodes
    private static String KEY_SUCCESS = "success";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        /**
         * Dashboard
         * */
        // Check login status in database
        userFunctions = new UserFunctions();
        if(userFunctions.isUserLoggedIn(getApplicationContext())){
            setContentView(R.layout.dashboard);

            /**
             * Se referencia a variables los controles del layout dashboard
             */
            btnOpen = (Button) findViewById(R.id.btnOpen);
```



```

        btnLogout = (Button) findViewById(R.id.btnLogout);
        spinnerRooms = (Spinner) findViewById(R.id.spinnerRooms);

        /**
         * Agregar las aulas disponibles al control Spinner -> Lista
         */

        List<String> roomsList = new ArrayList<String>();
        JSONObject jsonRooms = this.userFunctions.getRooms(); //Aulas
        disponibles en formato JSON

        int nRooms = jsonRooms.length(); //Cantidad de aulas disponibles

        for (int i=0; i<nRooms; i++){
            try {
                roomsList.add(jsonRooms.getString("aula " +
i));
            } catch (JSONException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

        //El adaptador permite cargar la lista de aulas al control
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
        android.R.layout.simple_spinner_item, roomsList);
        spinnerRooms.setAdapter(adapter);

        /**
         * Listener del botón ABRIR
         */
        btnOpen.setOnClickListener(new View.OnClickListener() {

            public void onClick(View view) {
                Log.d("Button", "Abrir");
                String room =
        spinnerRooms.getSelectedItem().toString();
                Log.d("ABRIR", room);
                JSONObject jsonRoom =
        userFunctions.openRoom(room);
                try {
                    Log.d("Abrir",
        jsonRoom.getString(KEY_SUCCESS));
                } catch (JSONException e1) {
                    // TODO Auto-generated catch block
                    e1.printStackTrace();
                }

                try {
                    if (jsonRoom.getString(KEY_SUCCESS) ==
"1"){
                        Context context =
        getApplicationContext();
                        CharSequence text = "LISTO!";
                        int duration = Toast.LENGTH_LONG;

                        Toast toast =
        Toast.makeText(context, text, duration);

```



```

                                toast.setGravity(Gravity.CENTER,
0, 150);
                                toast.show();
                                }
                                } catch (JSONException e) {
                                    // TODO Auto-generated catch block
                                    e.printStackTrace();
                                }
                            }
                        });

/**
 * Se crea el Listener del botón Logout
 */

btnLogout.setOnClickListener(new View.OnClickListener() {
    public void onClick(View arg0) {
        // TODO Auto-generated method stub
        userFunctions.logoutUser(getApplicationContext());
        Intent login = new Intent(getApplicationContext(),
LoginActivity.class);
        login.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(login);
        // Cerrar ventana del dashboard
        finish();
    }
});

}else{
    // El usuario no está logueado
    Intent login = new Intent(getApplicationContext(),
LoginActivity.class);
    login.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
    startActivity(login);
    // Cerrar el dashboard
    finish();
}

}
}

```

loginActivity.java

```

package org.app.tesis.activities;

import org.app.tesis.R;
import org.app.tesis.libraries.DatabaseHandler;
import org.app.tesis.libraries.UserFunctions;
import org.json.JSONException;
import org.json.JSONObject;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

```



```

public class LoginActivity extends Activity {
    Button btnLogin;
    Button btnLinkToRegister;
    EditText inputEmail;
    EditText inputPassword;
    TextView loginErrorMsg;

    // JSON Response node names
    private static String KEY_SUCCESS = "success";
    private static String KEY_ERROR = "error";
    private static String KEY_ERROR_MSG = "error_msg";
    private static String KEY_UID = "uid";
    private static String KEY_NAME = "name";
    private static String KEY_LASTNAME = "last_name";
    private static String KEY_CEDULA = "cedula";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.login);

        // Importing all assets like buttons, text fields
        inputEmail = (EditText) findViewById(R.id.loginCedula);
        inputPassword = (EditText) findViewById(R.id.loginPassword);
        btnLogin = (Button) findViewById(R.id.btnLogin);
        loginErrorMsg = (TextView) findViewById(R.id.login_error);

        // Login button Click Event
        btnLogin.setOnClickListener(new View.OnClickListener() {

            public void onClick(View view) {
                String email = inputEmail.getText().toString();
                String password = inputPassword.getText().toString();
                UserFunctions userFunction = new UserFunctions();
                Log.d("Button", "Login");
                JSONObject json = userFunction.loginUser(email,
password);

                // check for login response
                try {
                    if (json.getString(KEY_SUCCESS) != null) {
                        loginErrorMsg.setText("");
                        String res =
json.getString(KEY_SUCCESS);

                        if(Integer.parseInt(res) == 1){
                            // user successfully logged in
                            // Store user details in SQLite
                            Database
                            DatabaseHandler(getApplicationContext());

                            >Nombre y apellido

                            json.getJSONObject("teacher");

                            database

                            DatabaseHandler db = new
//Obtengo los datos del profesor-

                            JSONObject json_user =

                            // Clear all previous data in

```




```

        userFunction.logoutUser(getApplicationContext());

        db.addUser(json_user.getString(KEY_NAME), json_user.getString(KEY_LASTNAME),
        json.getString(KEY_UID), json.getString(KEY_CEDULA));

                                // Launch Dashboard Screen
                                Intent dashboard = new
Intent(getApplicationContext(), DashboardActivity.class);

                                // Close all views before
launching Dashboard

                                dashboard.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                                startActivity(dashboard);

                                // Close Login Screen
                                finish();
                                }else{
                                // Error in login
                                loginErrorMsg.setText("Verificar
CEDULA o PASSWORD");
                                }
                                } catch (JSONException e) {
                                e.printStackTrace();
                                }
                                }
                                });
        }
    }
}

```

DatabaseHandler.java

```

package org.app.thesis.libraries;

import java.util.HashMap;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class DatabaseHandler extends SQLiteOpenHelper {

    // All Static variables
    // Database Version
    private static final int DATABASE_VERSION = 1;

    // Database Name
    private static final String DATABASE_NAME = "android_api";

    // Login table name
    private static final String TABLE_LOGIN = "login";

    // Login Table Columns names
    private static final String KEY_ID = "id";
    private static final String KEY_NAME = "name";
    private static final String KEY_LASTNAME = "last_name";

```



```

private static final String KEY_UID = "uid";
private static final String KEY_CEDULA = "cedula";

public DatabaseHandler(Context context) {
    super(context, DATABASE_NAME, null, DATABASE_VERSION);
}

// Creating Tables
@Override
public void onCreate(SQLiteDatabase db) {
    String CREATE_LOGIN_TABLE = "CREATE TABLE " + TABLE_LOGIN + "("
        + KEY_ID + " INTEGER PRIMARY KEY,"
        + KEY_NAME + " TEXT,"
        + KEY_LASTNAME + " TEXT,"
        + KEY_UID + " TEXT,"
        + KEY_CEDULA + " TEXT" + ")";
    db.execSQL(CREATE_LOGIN_TABLE);
}

// Upgrading database
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    // Drop older table if existed
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_LOGIN);

    // Create tables again
    onCreate(db);
}

/**
 * Storing user details in database
 * */
public void addUser(String name, String last_name, String uid, String
cedula) {
    SQLiteDatabase db = this.getWritableDatabase();

    ContentValues values = new ContentValues();
    values.put(KEY_NAME, name); // Name
    values.put(KEY_LASTNAME, last_name); // Last name
    values.put(KEY_UID, uid); // UID
    values.put(KEY_CEDULA, cedula); // Cedula

    // Inserting Row
    db.insert(TABLE_LOGIN, null, values);
    db.close(); // Closing database connection
}

/**
 * Getting user data from database
 * */
public HashMap<String, String> getUserDetails(){
    HashMap<String,String> user = new HashMap<String,String>();
    String selectQuery = "SELECT * FROM " + TABLE_LOGIN;

    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery(selectQuery, null);
    // Move to first row
    cursor.moveToFirst();
    if(cursor.getCount() > 0){
        user.put("name", cursor.getString(1));
    }
}

```



```

        user.put("last_name", cursor.getString(2));
        user.put("uid", cursor.getString(3));
        user.put("cedula", cursor.getString(4));
    }
    cursor.close();
    db.close();
    // return user
    return user;
}

/**
 * Getting user login status
 * return true if rows are there in table
 * */
public int getRowCount() {
    String countQuery = "SELECT * FROM " + TABLE_LOGIN;
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery(countQuery, null);
    int rowCount = cursor.getCount();
    db.close();
    cursor.close();

    // return row count
    return rowCount;
}

/**
 * Re crate database
 * Delete all tables and create them again
 * */
public void resetTables(){
    SQLiteDatabase db = this.getWritableDatabase();
    // Delete All Rows
    db.delete(TABLE_LOGIN, null, null);
    db.close();
}
}

```

JSONParser.java

```

package org.app.thesis.libraries;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.UnsupportedEncodingException;
import java.util.List;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.json.JSONException;
import org.json.JSONObject;

```



```
import android.util.Log;

public class JSONParser {

    private static InputStream is = null;
    private static JSONObject jsonObj = null;
    private static JSONObject jsonObjRooms = null;
    private static String jsonRooms = "";
    private static String json = "";

    // Constructor
    public JSONParser() {
    }

    public JSONObject getJSONFromUrl(String url, List<NameValuePair> params) {
        /**
         * @param url
         * @param params
         *
         * Obtiene la respuesta en formato JSON enviada por la API.
         * Gets the response in JSON format from the API.
         */

        // Solicitud HTTP - HTTP Request
        try {
            // defaultHttpClient
            DefaultHttpClient httpClient = new DefaultHttpClient();
            //El metodo POST es aceptado por la API, GET es descartado
            HttpPost httpPost = new HttpPost(url);
            httpPost.setEntity(new UrlEncodedFormEntity(params));

            //Ejecutar la solicitud HTTP (POST), obtener la informacion
            HttpResponse httpResponse = httpClient.execute(httpPost);
            HttpEntity httpEntity = httpResponse.getEntity();
            is = httpEntity.getContent();

        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        } catch (ClientProtocolException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    try {
        BufferedReader reader = new BufferedReader(new
InputStreamReader(
                is, "iso-8859-1"), 8);
        StringBuilder sb = new StringBuilder();
        String line = null;
        while ((line = reader.readLine()) != null) {
            sb.append(line + "\n");
        }
        is.close();
        json = sb.toString();
        Log.e("JSON", json);
    } catch (Exception e) {
        Log.e("Buffer Error", "Error converting result " +
e.toString());
    }
}
```



```

    }

    // try parse the string to a JSON object
    try {
        jsonObj = new JSONObject(json);
    } catch (JSONException e) {
        Log.e("JSON Parser", "Error parsing data " + e.toString());
    }

    // return JSON String
    return jsonObj;
}

public JSONObject getJSONFromUrlGET(String url) {
    /**
     * @param url
     *
     * Obtiene la respuesta en formato JSON enviada por la API.
     * Gets the response in JSON format from the API.
     */

    // Solicitud HTTP - HTTP Request
    try {
        // defaultHttpClient
        DefaultHttpClient httpClient = new DefaultHttpClient();
        //El metodo GET es aceptado por la API
        HttpGet httpGet = new HttpGet(url);

        //Ejecutar la solicitud HTTP (GET), obtener la informacion
        HttpResponse httpResponse = httpClient.execute(httpGet);
        HttpEntity httpEntity = httpResponse.getEntity();
        is = httpEntity.getContent();

    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    } catch (ClientProtocolException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }

    try {
        InputStreamReader(
            new BufferedReader(new
                InputStreamReader(
                    is, "iso-8859-1"), 8);
            StringBuilder sb = new StringBuilder();
            String line = null;
            while ((line = reader.readLine()) != null) {
                sb.append(line + "\n");
            }
            is.close();
            jsonRooms = sb.toString();
            Log.e("JSON", jsonRooms);
        } catch (Exception e) {
            Log.e("Buffer Error", "Error converting result " +
                e.toString());
        }

        // try parse the string to a JSON object

```



```

        try {
            jsonObjRooms = new JSONObject(jsonRooms);
            Log.d("Parse", "aqui");
        } catch (JSONException e) {
            Log.e("JSON Parser", "Error parsing data " + e.toString());
            Log.d("Parse", "aca");
        }

        // return JSON String
        return jsonObjRooms;
    }
}

```

UserFunctions.java

```

package org.app.tesis.libraries;

import java.util.ArrayList;
import java.util.List;

import org.apache.http.NameValuePair;
import org.apache.http.message.BasicNameValuePair;
import org.json.JSONObject;

import android.content.Context;

public class UserFunctions {

    private JSONParser jsonParser;

    //private static String indexURL = "http://10.0.2.2/APItesis/index.php";
    localhost
    private static String indexURL = "http://192.168.1.114/APItesis/index.php";
    private static String getRoomsURL =
"http://192.168.1.114/APItesis/getRooms.php";
    private static String login_tag = "login";
    private static String open_tag = "open";

    // constructor
    public UserFunctions(){
        jsonParser = new JSONParser();
    }

    /**
     * function make Login Request
     * @param cedula
     * @param password
     */
    public JSONObject loginUser(String cedula, String password){
        // Building Parameters
        List<NameValuePair> params = new ArrayList<NameValuePair>();
        params.add(new BasicNameValuePair("tag", login_tag));
        params.add(new BasicNameValuePair("cedula", cedula));
        params.add(new BasicNameValuePair("password", password));
        JSONObject json = jsonParser.getJSONFromUrl(indexURL, params);
        // return json
    }
}

```



```
        // Log.e("JSON", json.toString());
        return json;
    }

    public JSONObject openRoom(String room){
        List<NameValuePair> params = new ArrayList<NameValuePair>();
        params.add(new BasicNameValuePair("tag", open_tag));
        params.add(new BasicNameValuePair("room", room));
        JSONObject jsonRoom = jsonParser.getJSONFromUrl(indexURL, params);
        return jsonRoom;
    }

    public JSONObject getRooms(){
        JSONObject jsonRooms = jsonParser.getJSONFromUrlGET(getRoomsURL);
        return jsonRooms;
    }

    /**
     * Function get Login status
     */
    public boolean isUserLoggedIn(Context context){
        DatabaseHandler db = new DatabaseHandler(context);
        int count = db.getRowCount();
        if(count > 0){
            // user logged in
            return true;
        }
        return false;
    }

    /**
     * Function to logout user
     * Reset Database
     */
    public boolean logoutUser(Context context){
        DatabaseHandler db = new DatabaseHandler(context);
        db.resetTables();
        return true;
    }
}
```